Red Hat OpenShift Container Storage 4.8

Managing hybrid and multicloud resources

클러스터 및 스토리지 관리자를 위한 하이브리드 및 다중 클라우드 리소스 관리

Last Updated: 2023-04-08

Red Hat OpenShift Container Storage 4.8 Managing hybrid and multicloud resources

클러스터 및 스토리지 관리자를 위한 하이브리드 및 다중 클라우드 리소스 관리

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Managing_hybrid_and_multicloud_resources.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

이 문서에서는 하이브리드 클라우드 또는 다중 클라우드 환경에서 스토리지 리소스를 관리하는 방법을 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체 4

RED HAT 문서에 관한 피드백 제공 5

1장. MULTICLOUD OBJECT GATEWAY 정보 6

2장. 애플리케이션을 사용하여 MULTICLOUD OBJECT GATEWAY에 액세스 7
2.1. 터미널에서 MULTICLOUD OBJECT GATEWAY에 액세스 8
2.2. MCG 명령줄 인터페이스에서 MULTICLOUD OBJECT GATEWAY에 액세스 10

3장. MULTICLOUD OBJECT GATEWAY CONSOLE에 대한 사용자 액세스 허용 14

4장. 하이브리드 또는 MULTICLOUD 용 스토리지 리소스 추가 16
4.1. 새 백업 저장소 생성 16
4.2. MCG 명령줄 인터페이스를 사용하여 하이브리드 또는 MULTICLOUD용 스토리지 리소스 추가 18
4.2.1. AWS 지원 백업 저장소 생성 19
4.2.2. IBM COS 지원 백업 저장소 생성 21
4.2.3. Azure 지원 백업 저장소 생성 23
4.2.4. GCP 지원 백업 저장소 생성 26
4.2.5. 로컬 영구 볼륨 백업 백업 저장소 생성 28
4.3. S3 호환 MULTICLOUD OBJECT GATEWAY 백업 저장소 생성 30
4.4. 사용자 인터페이스를 사용하여 하이브리드 및 MULTICLOUD용 스토리지 리소스 추가 32
4.5. 새 버킷 클래스 생성 34
4.6. 버킷 클래스 편집 36
4.7. 버킷 클래스의 백업 저장소 편집 37

5장. 네임스페이스 버킷 관리 40
5.1. 네임스페이스 버킷의 오브젝트에 대한 AMAZON S3 API 끝점 40
5.2. MULTICLOUD OBJECT GATEWAY CLI 및 YAML을 사용하여 네임스페이스 버킷 추가 41
5.2.1. YAML을 사용하여 AWS S3 네임 스페이스 버킷 추가 42
5.2.2. YAML을 사용하여 IBM COS 네임 스페이스 버킷 추가 45
5.2.3. Multicloud Object Gateway CLI를 사용하여 AWS S3 네임 스페이스 버킷 추가 48
5.2.4. Multicloud Object Gateway CLI를 사용하여 IBM COS 네임 스페이스 버킷 추가 51
5.3. OPENSHIFT CONTAINER PLATFORM 사용자 인터페이스를 사용하여 네임스페이스 버킷 추가 54

6장. 하이브리드 및 멀티 클라우드 버킷의 데이터 미러링 58
6.1. MCG 명령줄 인터페이스를 사용하여 데이터를 미러링하는 버킷 클래스 생성 58
6.2. YAML을 사용하여 데이터를 미러링하도록 버킷 클래스 생성 58
6.3. 사용자 인터페이스를 사용하여 데이터를 미러링하도록 버킷 구성 59

7장. MULTICLOUD OBJECT GATEWAY의 버킷 정책 61
7.1. 버킷 정책 정보 61
7.2. 버킷 정책 사용 61
7.3. MULTICLOUD OBJECT GATEWAY에서 AWS S3 사용자 생성 63

8장. 개체 버킷 클레임 67
8.1. 동적 개체 버킷 클레임 67
8.2. 명령줄 인터페이스를 사용하여 개체 버킷 클레임 생성 70
8.3. OPENSHIFT 웹 콘솔을 사용하여 개체 버킷 클레임 생성 74
8.4. 배포에 오브젝트 버킷 클레임 연결 77
8.5. OPENSHIFT 웹 콘솔을 사용하여 오브젝트 버킷 보기 78
8.6. 개체 버킷 클레임 삭제 79

9장. 오브젝트 버킷의 캐싱 정책 82

9.1. AWS 캐시 버킷 생성 82
9.2. IBM COS 캐시 버킷 생성 85

10장. 끝점을 추가하여 MULTICLOUD OBJECT GATEWAY 성능 확장 89
10.1. MULTICLOUD OBJECT GATEWAY의 S3 끝점 89
10.2. 스토리지 노드를 사용한 확장 89

11장. MULTICLOUD OBJECT GATEWAY 엔드 포인트 자동 확장 93

12장. RADOS 오브젝트 게이트웨이 S3 끝점 액세스 94

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. Red Hat이 이를 개선하는 방법을 알려 주십시오. 피드백을 제공하려면 다음을 수행하십시오.

- 특정 문구에 대한 간단한 주석은 다음과 같습니다.
  1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **Feedback** (피드백) 버튼이 있는지 확인합니다.
  2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
  3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
  4. 표시된 지침을 따릅니다.
- 보다 상세하게 피드백을 제출하려면 다음과 같이 Bugzilla 티켓을 생성하십시오.
  1. Bugzilla 웹 사이트로 이동하십시오.
  2. Component로 **Documentation**을 선택하십시오.
  3. **Description** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
  4. **Submit Bug**를 클릭하십시오.

1장. MULTICLOUD OBJECT GATEWAY 정보

MCG(Multicloud Object Gateway)는 OpenShift용 경량 오브젝트 스토리지 서비스로, 사용자가 소규모를 시작한 다음 필요에 따라 온프레미스, 여러 클러스터에서 클라우드 네이티브 스토리지를 사용하여 확장할 수 있습니다.

2장. 애플리케이션을 사용하여 MULTICLOUD OBJECT GATEWAY에 액세스

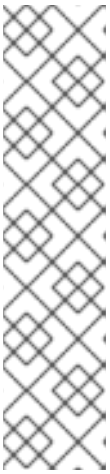
AWS S3을 대상으로 하는 모든 애플리케이션 또는 AWS S3 Software Development Kit(SDK)를 사용하는 코드를 사용하여 오브젝트 서비스에 액세스할 수 있습니다. 애플리케이션은 MCG 끝점, 액세스 키 및 시크릿 액세스 키를 지정해야 합니다. 터미널 또는 MCG CLI를 사용하여 이 정보를 검색할 수 있습니다.

RADOS Object Gateway S3 끝점 액세스에 대한 자세한 내용은 12장. *RADOS 오브젝트 게이트웨이 S3 끝점 액세스* 을 참조하십시오.

사전 요구 사항

- 실행 중인 OpenShift Container Storage Platform
- 더 쉽게 관리 할 수 있도록 MCG 명령줄 인터페이스를 다운로드하십시오.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```



참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

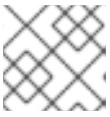
- IBM Power Systems의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- IBM Z 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 Download RedHat OpenShift Container Storage(**RedHat OpenShift Container Storage 다운로드) 페이지에 있는 OpenShift Container Storage RPM에서 mcg** 패키지를 설치할 수도 있습니다.



참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

다음 두 가지 방법으로 관련 끝점, 액세스 키 및 시크릿 액세스 키에 액세스할 수 있습니다.

- 2.1절. "터미널에서 Multicloud Object Gateway에 액세스"
- 2.2절. "MCG 명령줄 인터페이스에서 Multicloud Object Gateway에 액세스"

가상 호스팅 스타일을 사용하여 MCG 버킷 액세스

예 2.1. 예제

클라이언트 애플리케이션이 https://<bucket-name>.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com에 액세스하려고 하는 경우

> 여기서 **<bucket-name>** 은 MCG 버킷의 이름입니다.
>
> 예: https://mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
>
> **mcg-test-bucket.s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com** 에서 S3 서비스를 가리키려면 DNS 항목이 필요합니다.

중요

클라이언트 애플리케이션이 가상 호스팅 스타일을 사용하여 MCG 버킷을 가리키도록 DNS 항목이 있는지 확인합니다.

2.1. 터미널에서 MULTICLOUD OBJECT GATEWAY에 액세스

절차

describe 명령을 실행하여 액세스 키(**AWS_ACCESS_KEY_ID 값) 및 시크릿 액세스 키 (AWS_SECRET_ACCESS_KEY** 값)를 포함하여 MCG 끝점에 대한 정보를 봅니다.

```
# oc describe noobaa -n openshift-storage
```

출력은 다음과 유사합니다.

```
Name:         noobaa
Namespace:    openshift-storage
Labels:       <none>
Annotations:  <none>
API Version:  noobaa.io/v1alpha1
Kind:         NooBaa
Metadata:
  Creation Timestamp:  2019-07-29T16:22:06Z
  Generation:          1
  Resource Version:    6718822
  Self Link:           /apis/noobaa.io/v1alpha1/namespaces/openshift-storage/noobaas/noobaa
  UID:                 019cfb4a-b21d-11e9-9a02-06c8de012f9e
Spec:
Status:
  Accounts:
    Admin:
      Secret Ref:
        Name:          noobaa-admin
        Namespace:     openshift-storage
  Actual Image:        noobaa/noobaa-core:4.0
  Observed Generation:  1
  Phase:               Ready
  Readme:

  Welcome to NooBaa!
  -----------------
```

```
Welcome to NooBaa!
  ----------------
  NooBaa Core Version:
  NooBaa Operator Version:

  Lets get started:

  1. Connect to Management console:

    Read your mgmt console login information (email & password) from secret: "noobaa-admin".

      kubectl get secret noobaa-admin -n openshift-storage -o json | jq '.data|map_values(@base64d)'

    Open the management console service - take External IP/DNS or Node Port or use port
forwarding:

      kubectl port-forward -n openshift-storage service/noobaa-mgmt 11443:443 &
      open https://localhost:11443

  2. Test S3 client:

    kubectl port-forward -n openshift-storage service/s3 10443:443 &
(1)
    NOOBAA_ACCESS_KEY=$(kubectl get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_ACCESS_KEY_ID|@base64d')
(2)
    NOOBAA_SECRET_KEY=$(kubectl get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_SECRET_ACCESS_KEY|@base64d')
    alias s3='AWS_ACCESS_KEY_ID=$NOOBAA_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=$NOOBAA_SECRET_KEY aws --endpoint https://localhost:10443 --
no-verify-ssl s3'
    s3 ls


  Services:
    Service Mgmt:
      External DNS:
        https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
        https://a3406079515be11eaa3b70683061451e-1194613580.us-east-
2.elb.amazonaws.com:443
      Internal DNS:
        https://noobaa-mgmt.openshift-storage.svc:443
      Internal IP:
        https://172.30.235.12:443
      Node Ports:
        https://10.0.142.103:31385
      Pod Ports:
        https://10.131.0.19:8443
    serviceS3:
      External DNS: (3)
        https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
        https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443
      Internal DNS:
        https://s3.openshift-storage.svc:443
      Internal IP:
        https://172.30.86.41:443
```

```
Node Ports:
  https://10.0.142.103:31011
Pod Ports:
  https://10.131.0.19:6443
```

1

액세스 키 (**AWS_ACCESS_KEY_ID** 값)

2

시크릿 액세스 키 (**AWS_SECRET_ACCESS_KEY** 값)

3

MCG 엔드 포인트

참고

oc describe noobaa 명령의 출력에는 사용 가능한 내부 및 외부 **DNS** 이름이 나열됩니다. 내부 **DNS**를 사용하는 경우 트래픽은 사용 가능합니다. 외부 **DNS**는 **Load Balancing**을 사용하여 트래픽을 처리하므로 시간당 비용이 있습니다.

2.2. MCG 명령줄 인터페이스에서 MULTICLOUD OBJECT GATEWAY에 액세스

사전 요구 사항

- **MCG** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

- **IBM Power Systems**의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

절차

status 명령을 실행하여 엔드포인트, 액세스 키 및 시크릿 액세스 키에 액세스합니다.

```
noobaa status -n openshift-storage
```

출력은 다음과 유사합니다.

```
INFO[0000] Namespace: openshift-storage
INFO[0000]
INFO[0000] CRD Status:
INFO[0003]   Exists: CustomResourceDefinition "noobaas.noobaa.io"
INFO[0003]   Exists: CustomResourceDefinition "backingstores.noobaa.io"
INFO[0003]   Exists: CustomResourceDefinition "bucketclasses.noobaa.io"
INFO[0004]   Exists: CustomResourceDefinition "objectbucketclaims.objectbucket.io"
INFO[0004]   Exists: CustomResourceDefinition "objectbuckets.objectbucket.io"
INFO[0004]
INFO[0004] Operator Status:
INFO[0004]   Exists: Namespace "openshift-storage"
INFO[0004]   Exists: ServiceAccount "noobaa"
INFO[0005]   Exists: Role "ocs-operator.v0.0.271-6g45f"
INFO[0005]   Exists: RoleBinding "ocs-operator.v0.0.271-6g45f-noobaa-f9vpj"
INFO[0006]   Exists: ClusterRole "ocs-operator.v0.0.271-fjhgh"
INFO[0006]   Exists: ClusterRoleBinding "ocs-operator.v0.0.271-fjhgh-noobaa-pdxn5"
INFO[0006]   Exists: Deployment "noobaa-operator"
INFO[0006]
INFO[0006] System Status:
INFO[0007]   Exists: NooBaa "noobaa"
INFO[0007]   Exists: StatefulSet "noobaa-core"
INFO[0007]   Exists: Service "noobaa-mgmt"
INFO[0008]   Exists: Service "s3"
```

```
INFO[0008]   Exists: Secret "noobaa-server"
INFO[0008]   Exists: Secret "noobaa-operator"
INFO[0008]   Exists: Secret "noobaa-admin"
INFO[0009]   Exists: StorageClass "openshift-storage.noobaa.io"
INFO[0009]   Exists: BucketClass "noobaa-default-bucket-class"
INFO[0009]   (Optional) Exists: BackingStore "noobaa-default-backing-store"
INFO[0010]   (Optional) Exists: CredentialsRequest "noobaa-cloud-creds"
INFO[0010]   (Optional) Exists: PrometheusRule "noobaa-prometheus-rules"
INFO[0010]   (Optional) Exists: ServiceMonitor "noobaa-service-monitor"
INFO[0011]   (Optional) Exists: Route "noobaa-mgmt"
INFO[0011]   (Optional) Exists: Route "s3"
INFO[0011]   Exists: PersistentVolumeClaim "db-noobaa-core-0"
INFO[0011]   System Phase is "Ready"
INFO[0011]   Exists:  "noobaa-admin"

#------------------#
#- Mgmt Addresses -#
#------------------#

ExternalDNS : [https://noobaa-mgmt-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a3406079515be11eaa3b70683061451e-1194613580.us-east-2.elb.amazonaws.com:443]
ExternalIP  : []
NodePorts   : [https://10.0.142.103:31385]
InternalDNS : [https://noobaa-mgmt.openshift-storage.svc:443]
InternalIP  : [https://172.30.235.12:443]
PodPorts    : [https://10.131.0.19:8443]

#--------------------#
#- Mgmt Credentials -#
#--------------------#

email    : admin@noobaa.io
password : HKLbH1rSuVU0I/souIkSiA==

#----------------#
#- S3 Addresses -#
#----------------#

(1)
ExternalDNS : [https://s3-openshift-storage.apps.mycluster-cluster.qe.rh-ocs.com
https://a340f4e1315be11eaa3b70683061451e-943168195.us-east-2.elb.amazonaws.com:443]
ExternalIP  : []
NodePorts   : [https://10.0.142.103:31011]
InternalDNS : [https://s3.openshift-storage.svc:443]
InternalIP  : [https://172.30.86.41:443]
PodPorts    : [https://10.131.0.19:6443]

#------------------#
#- S3 Credentials -#
#------------------#

(2)
AWS_ACCESS_KEY_ID     : jVmAsu9FsvRHYmfjTiHV
(3)
AWS_SECRET_ACCESS_KEY : E//420VNedJfATvVSmDz6FMtsSAzuBv6z180PT5c
```

```
#-----------------#
#- Backing Stores -#
#-----------------#

NAME                          TYPE     TARGET-BUCKET                                      PHASE   AGE
noobaa-default-backing-store   aws-s3   noobaa-backing-store-15dc896d-7fe0-4bed-9349-
5942211b93c9   Ready   141h35m32s

#------------------#
#- Bucket Classes -#
#------------------#

NAME                          PLACEMENT                                                  PHASE   AGE
noobaa-default-bucket-class   {Tiers:[{Placement: BackingStores:[noobaa-default-backing-store]}]}
Ready   141h35m33s

#----------------#
#- Bucket Claims -#
#----------------#

No OBC's found.
```

1

엔드포인트

2

액세스 키

3

시크릿 액세스 키

이제 애플리케이션에 연결하기 위해 관련 끝점, 액세스 키 및 비밀 액세스 키가 있어야 합니다.

예 2.2. 예제

AWS S3 CLI가 애플리케이션인 경우 다음 명령은 **OpenShift Container Storage**의 버킷을 나열합니다.

```
AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY_ID>
AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KEY>
aws --endpoint <ENDPOINT> --no-verify-ssl s3 ls
```

3장. MULTICLOUD OBJECT GATEWAY CONSOLE에 대한 사용자 액세스 허용

사용자에게 **Multicloud Object Gateway Console**에 액세스할 수 있도록 하려면 사용자가 다음 조건을 충족하는지 확인합니다.

- 사용자는 **cluster-admins** 그룹에 있습니다.
- 사용자는 **system:cluster-admins** 가상 그룹에 있습니다.

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform.**

절차

1. **Multicloud Object Gateway** 콘솔에 대한 액세스를 활성화합니다.

   클러스터에서 다음 단계를 한 번 수행합니다.

   a. **cluster-admins** 그룹을 만듭니다.

   ```
   # oc adm groups new cluster-admins
   ```

   b. 그룹을 **cluster-admin** 역할에 바인딩합니다.

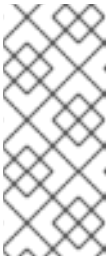
   ```
   # oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
   ```

2. **cluster-admins** 그룹에서 사용자를 추가하거나 제거하여 **Multicloud Object Gateway** 콘솔에 대한 액세스를 제어합니다.

   - **cluster-admins** 그룹에 사용자를 추가하려면 다음을 수행합니다.

   ```
   # oc adm groups add-users cluster-admins <user-name> <user-name> <user-name>...
   ```

여기서 **<user-name>** 은 추가할 사용자의 이름입니다.



참고

cluster-admins 그룹에 사용자 세트를 추가하는 경우 **OpenShift Container Storage** 대시보드에 액세스할 수 있도록 새로 추가된 사용자를 **cluster-admin** 역할에 바인딩할 필요가 없습니다.

- **cluster-admins** 그룹에서 사용자를 제거하려면 다음을 수행합니다.

```
# oc adm groups remove-users cluster-admins <user-name> <user-name> <user-name>...
```

여기서 **<user-name>** 은 제거할 사용자의 이름입니다.

검증 단계

1. **OpenShift** 웹 콘솔에서 **Multicloud Object Gateway Console**에 대한 액세스 권한이 있는 사용자로 로그인합니다.
2. 스토리지 → 개요 → 오브젝트 탭 → **Multicloud Object Gateway** 링크를 선택합니다.
3. **Multicloud Object Gateway Console**에서 액세스 권한이 있는 동일한 사용자로 로그인합니다.
4. **Allow selected permissions** (선택한 권한 허용)를 클릭합니다.

4장. 하이브리드 또는 MULTICLOUD 용 스토리지 리소스 추가

4.1. 새 백업 저장소 생성

OpenShift Container Storage에서 새 백업 저장소를 생성하려면 다음 절차를 사용하십시오.

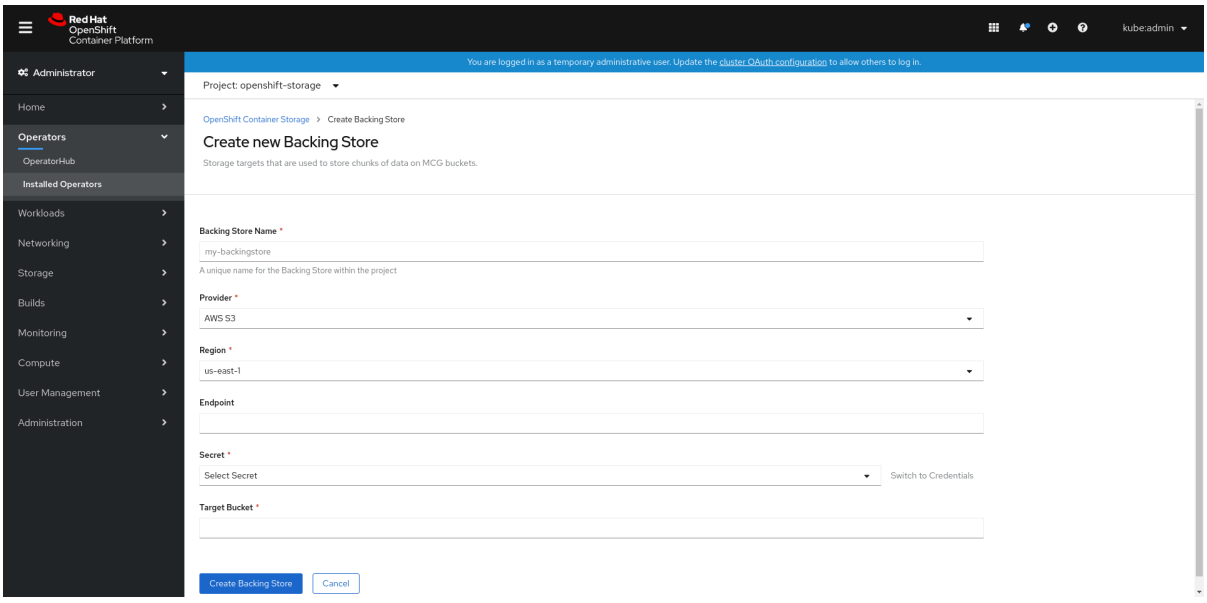
사전 요구 사항

- **OpenShift**에 대한 관리자 액세스.

절차

1. **OpenShift** 웹 콘솔의 왼쪽 창에서 **Operators** → 설치된 **Operator** 를 클릭하여 설치된 **Operator**를 확인합니다.
2. **OpenShift Container Storage Operator**를 클릭합니다.
3. **OpenShift Container Storage Operator** 페이지에서 오른쪽 스크롤하고 백업 저장소 탭을 클릭합니다.
4. **Create Backing Store** (백업 저장소 만들기)를 클릭합니다.

그림 4.1. 백업 저장소 페이지 만들기



5. **Create New Backing Store**(새 백업 저장소 생성) 페이지에서 다음을 수행합니다.

a. 백업 저장소 이름을 입력합니다.

b. 공급업체 선택.

c. 지역 선택.

d. 엔드포인트 입력. 이는 선택 사항입니다.

e. 드롭다운 목록에서 **Secret** (시크릿)을 선택하거나 고유한 보안을 생성합니다. 필요한 경우 **Credentials**(자격 증명) 보기로 전환 하여 필요한 시크릿을 입력할 수 있습니다.

OCP 보안 생성에 대한 자세한 내용은 **Openshift Container Platform** 설명서의 보안 생성 섹션을 참조하십시오.

각 백업 저장소에는 다른 보안이 필요합니다. 특정 백업 저장소에 대한 시크릿 생성에 대한 자세한 내용은 **4.2**절. **"MCG** 명령줄 인터페이스를 사용하여 하이브리드 또는 **Multicloud** 용 스토리지 리소스 추가" 을 참조하고 **YAML**을 사용하여 스토리지 리소스 추가 절차를 따르십시오.

참고

이 메뉴는 **Google Cloud** 및 로컬 **PVC**를 제외한 모든 공급자와 관련이 있습니다.

f. 대상 버킷을 입력합니다. 대상 버킷은 원격 클라우드 서비스에서 호스팅되는 컨테이너 스토리지입니다. **MCG**가 시스템에 이 버킷을 사용할 수 있음을 알려주는 연결을 만들 수 있습니다.

6. **Create Backing Store** (백업 저장소 만들기)를 클릭합니다.

검증 단계

1. **Operators** → 설치된 **Operators**를 클릭합니다.

2. **OpenShift Container Storage Operator**를 클릭합니다.

3. 새 백업 저장소를 검색하거나 백업 저장소 탭을 클릭하여 모든 백업 저장소를 확인합니다.

4.2. MCG 명령줄 인터페이스를 사용하여 하이브리드 또는 MULTICLOUD용 스토리지 리소스 추가

MCG(Multicloud Object Gateway)는 클라우드 공급자와 클러스터 전체에서 데이터를 포괄하는 프로세스를 간소화합니다.

MCG에서 사용할 수 있는 백업 스토리지를 추가해야 합니다.

배포 유형에 따라 다음 절차 중 하나를 선택하여 백업 스토리지를 생성할 수 있습니다.

- **AWS** 지원 백업 저장소를 생성하려면 를 참조하십시오. 4.2.1절. “AWS 지원 백업 저장소 생성”
- **IBM COS** 지원 백업 저장소를 생성하려면 다음을 참조하십시오. 4.2.2절. “IBM COS 지원 백업 저장소 생성”
- **Azure** 지원 백업 저장소를 생성하려면 를 참조하십시오. 4.2.3절. “Azure 지원 백업 저장소 생성”
- **GCP** 백업 백업 저장소를 생성하려면 를 참조하십시오. 4.2.4절. “GCP 지원 백업 저장소 생성”
- 로컬 영구 볼륨 지원 백업 저장소를 생성하려면 다음을 참조하십시오. 4.2.5절. “로컬 영구 볼륨 백업 백업 저장소 생성”

VMware 배포의 경우 4.3절. “s3 호환 Multicloud Object Gateway 백업 저장소 생성” 로 건너뜁니다.

4.2.1. AWS 지원 백업 저장소 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages**에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.

참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa backingstore create aws-s3 <backingstore_name> --access-key=<AWS ACCESS KEY> --secret-key=<AWS SECRET ACCESS KEY> --target-bucket <bucket-name> -n openshift-storage
```

a. **<backingstore_name>** 을 백업 저장소의 이름으로 바꿉니다.

b. **<AWS ACCESS KEY>** 및 **<AWS SECRET ACCESS KEY>** 를 이 목적을 위해 생성한 **AWS** 액세스 키 **ID** 및 시크릿 액세스 키로 바꿉니다.

c. **<bucket-name>** 을 기존 **AWS** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object**

Gateway에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

출력은 다음과 유사합니다.

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "aws-resource"
INFO[0002]   Created: Secret "backing-store-secret-aws-resource"
```

YAML을 사용하여 스토리지 리소스를 추가할 수도 있습니다.

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
  namespace: openshift-storage
type: Opaque
data:
  AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
  AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
```

   a. **Base64**를 사용하여 자체 **AWS** 액세스 키 **ID** 및 시크릿 액세스 키를 제공하고 인코딩해야 하며 **<AWS ACCESS KEY ID ENCODED in BASE64>** 및 **< AWS SECRET ACCESS KEY ENCODED in BASE64>** 대신 결과를 사용해야 합니다.

   b. **<backingstore-secret-name>** 을 고유한 이름으로 바꿉니다.

2. 특정 백업 저장소에 대해 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  awsS3:
    secret:
```

```
    name: <backingstore-secret-name>
    namespace: openshift-storage
  targetBucket: <bucket-name>
type: aws-s3
```

a. **<bucket-name>** 을 기존 **AWS** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

b. **<backingstore-secret-name>** 을 이전 단계에서 생성한 보안의 이름으로 바꿉니다.

4.2.2. IBM COS 지원 백업 저장소 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

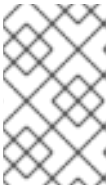
- **IBM Power Systems**의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages**에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.



참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. MCG 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa backingstore create ibm-cos <backingstore_name> --access-key=<IBM ACCESS KEY> --secret-key=<IBM SECRET ACCESS KEY> --endpoint=<IBM COS ENDPOINT> --target-bucket <bucket-name> -n openshift-storage
```

a. <backingstore_name> 을 백업 저장소의 이름으로 바꿉니다.

b. <IBM ACCESS KEY>, <IBM SECRET ACCESS KEY>, <IBM COS ENDPOINT> 를 IBM 액세스 키 ID, 시크릿 액세스 키 및 기존 IBM 버킷 위치에 해당하는 해당 지역 엔드포인트로 바꿉니다.

IBM 클라우드에서 위의 키를 생성하려면 대상 버킷에 대한 서비스 자격 증명을 생성하는 동안 HMAC 인증 정보를 포함해야 합니다.

c. <bucket-name> 을 기존 IBM 버킷 이름으로 바꿉니다. 이 인수는 Multicloud Object Gateway에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

출력은 다음과 유사합니다.

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "ibm-resource"
INFO[0002]   Created: Secret "backing-store-secret-ibm-resource"
```

YAML을 사용하여 스토리지 리소스를 추가할 수도 있습니다.

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
 name: <backingstore-secret-name>
```

```
type: Opaque
data:
  IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
  IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED IN
BASE64>
```

a. **Base64**를 사용하여 자체 **IBM COS** 액세스 키 **ID** 및 비밀 액세스 키를 제공하고 인코딩해야 하며 **<IBM COS COS** 키 **ID ENCODED IN BASE64>** 및 **< IBM COS SECRET ACCESS KEY ENCODED IN BASE64>** 대신 결과를 사용해야 합니다.

b. **<backingstore-secret-name>** 을 고유한 이름으로 바꿉니다.

2. 특정 백업 저장소에 대해 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  ibmCos:
    endpoint: <endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBucket: <bucket-name>
  type: ibm-cos
```

a. **<bucket-name>** 을 기존 **IBM COS** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

b. **<endpoint>** 를 기존 **IBM** 버킷 이름의 위치에 해당하는 지역 끝점으로 바꿉니다. 이 인수는 백업 저장소에 사용할 엔드포인트를 **Multicloud Object Gateway**에 지시하고 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

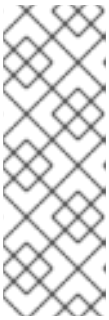
c. **<backingstore-secret-name>** 을 이전 단계에서 생성한 보안의 이름으로 바꿉니다.

4.2.3. Azure 지원 백업 저장소 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

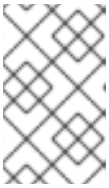


참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages**에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.



참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa backingstore create azure-blob <backingstore_name> --account-key=<AZURE ACCOUNT KEY> --account-name=<AZURE ACCOUNT NAME> --target-blob-container <blob container name>
```

   a. **<backingstore_name>** 을 백업 저장소의 이름으로 바꿉니다.

   b. **<AZURE ACCOUNT KEY>** 및 **<AZURE ACCOUNT NAME>** 을 이 목적을 위해 생성한 **AZURE** 계정 키 및 계정 이름으로 바꿉니다.

   c. **<blob** 컨테이너 이름**>** 을 기존 **Azure Blob** 컨테이너 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

출력은 다음과 유사합니다.

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "azure-resource"
INFO[0002]   Created: Secret "backing-store-secret-azure-resource"
```

YAML을 사용하여 스토리지 리소스를 추가할 수도 있습니다.

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
type: Opaque
data:
  AccountName: <AZURE ACCOUNT NAME ENCODED IN BASE64>
  AccountKey: <AZURE ACCOUNT KEY ENCODED IN BASE64>
```

   a. **Base64**를 사용하여 자체 **Azure** 계정 이름과 계정 키를 제공하고 인코딩해야 하며 **<AZURE ACCOUNT NAME ENCODED IN BASE64>** 및 **< AZURE ACCOUNT KEY ENCODED IN BASE64>** 대신 결과를 사용해야 합니다.

   b. **<backingstore-secret-name>** 을 고유한 이름으로 바꿉니다.

2. 특정 백업 저장소에 대해 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  azureBlob:
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBlobContainer: <blob-container-name>
  type: azure-blob
```

a. **<blob-container-name>** 을 기존 **Azure Blob** 컨테이너 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

b. **<backingstore-secret-name>** 을 이전 단계에서 생성한 보안의 이름으로 바꿉니다.

4.2.4. GCP 지원 백업 저장소 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages**에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.

참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa backingstore create google-cloud-storage <backingstore_name> --private-key-json-file=<PATH TO GCP PRIVATE KEY JSON FILE> --target-bucket <GCP bucket name>
```

a. **<backingstore_name>** 을 백업 저장소의 이름으로 바꿉니다.

b. **<PATH TO GCP PRIVATE KEY JSON FILE>** 을 이 목적을 위해 생성된 **GCP** 개인 키의 경로로 바꿉니다.

c. **<GCP** 버킷 이름**>** 을 기존 **GCP** 오브젝트 스토리지 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

출력은 다음과 유사합니다.

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Created: BackingStore "google-gcp"
INFO[0002]   Created: Secret "backing-store-google-cloud-storage-gcp"
```

YAML을 사용하여 스토리지 리소스를 추가할 수도 있습니다.

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: <backingstore-secret-name>
type: Opaque
data:
  GoogleServiceAccountPrivateKeyJson: <GCP PRIVATE KEY ENCODED IN BASE64>
```

a. **Base64**를 사용하여 자체 **GCP** 서비스 계정 개인 키를 제공하고 인코딩해야 하며 **<GCP PRIVATE KEY ENCODED IN BASE64>** 대신 결과를 사용해야 합니다.

b. **<backingstore-secret-name>**을 고유한 이름으로 바꿉니다.

2. 특정 백업 저장소에 대해 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
```

```
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  googleCloudStorage:
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    targetBucket: <target bucket>
  type: google-cloud-storage
```

a. **<target bucket>** 을 기존 **Google** 스토리지 버킷으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

b. **<backingstore-secret-name>** 을 이전 단계에서 생성한 보안의 이름으로 바꿉니다.

4.2.5. 로컬 영구 볼륨 백업 백업 저장소 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

- 또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/packages**에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.

참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. **MCG 명령줄 인터페이스에서 다음 명령을 실행합니다.**

```
noobaa backingstore create  pv-pool <backingstore_name> --num-volumes=<NUMBER OF VOLUMES>  --pv-size-gb=<VOLUME SIZE> --storage-class=<LOCAL STORAGE CLASS>
```

   a. **<backingstore_name> 을 백업 저장소의 이름으로 바꿉니다.**

   b. **<NUMBER OF VOLUMES> 를 생성하려는 볼륨 수로 바꿉니다. 볼륨 수를 늘리면 스토리지가 확장됩니다.**

   c. **<VOLUME SIZE> 를 각 볼륨의 필수 크기 (GB)로 바꿉니다.**

   d. **ocs-storagecluster-ceph-rbd를 사용하는 데 권장되는 로컬 스토리지 클래스로 <LOCAL STORAGE CLASS> 를 바꿉니다.**

   출력은 다음과 유사합니다.

```
INFO[0001]   Exists: NooBaa "noobaa"
INFO[0002]   Exists: BackingStore "local-mcg-storage"
```

YAML을 사용하여 스토리지 리소스를 추가할 수도 있습니다.

1. **특정 백업 저장소에 대해 다음 YAML을 적용합니다.**

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <backingstore_name>
  namespace: openshift-storage
spec:
   pvPool:
    numVolumes: <NUMBER OF VOLUMES>
    resources:
      requests:
```

```
      storage: <VOLUME SIZE>
    storageClass: <LOCAL STORAGE CLASS>
  type: pv-pool
```

a. **<backingstore_name>** 을 백업 저장소의 이름으로 바꿉니다.

b. **<NUMBER OF VOLUMES>** 를 생성하려는 볼륨 수로 바꿉니다. 볼륨 수를 늘리면 스토리지가 확장됩니다.

c. **<VOLUME SIZE>** 를 각 볼륨의 필수 크기(GB)로 바꿉니다. 문자 G는 유지되어야 합니다

d. **ocs-storagecluster-ceph-rbd**를 사용하는 데 권장되는 로컬 스토리지 클래스로 **<LOCAL STORAGE CLASS>** 를 바꿉니다.

4.3. S3 호환 MULTICLOUD OBJECT GATEWAY 백업 저장소 생성

Multicloud Object Gateway는 **S3** 호환 오브젝트 스토리지를 백업 저장소(예: **Red Hat Ceph Storage**의 **RADOS Gateway**)로 사용할 수 있습니다. 다음 절차에서는 **Red Hat Ceph Storage**의 **RADOS** 게이트웨이에 대해 **S3** 호환 **Multicloud Object Gateway** 백업 저장소를 생성하는 방법을 보여줍니다. **RGW**를 배포할 때 **Openshift Container Storage Operator**는 **Multicloud Object Gateway**에 대한 **S3** 호환 백업 저장소를 자동으로 생성합니다.

절차

1. **MCG(Multicloud Object Gateway)** 명령줄 인터페이스에서 다음 **NooBaa** 명령을 실행합니다.

```
noobaa backingstore create s3-compatible rgw-resource --access-key=<RGW ACCESS
KEY> --secret-key=<RGW SECRET KEY> --target-bucket=<bucket-name> --endpoint=
<RGW endpoint>
```

a. **<RGW ACCESS KEY>** 및 **<RGW SECRET KEY>** 를 가져오려면 **RGW** 사용자 시크릿 이름을 사용하여 다음 명령을 실행합니다.

```
oc get secret <RGW USER SECRET NAME> -o yaml -n openshift-storage
```

b. **Base64**에서 액세스 키 **ID**와 액세스 키를 디코딩하고 유지합니다.

c.

<RGW USER ACCESS KEY> 및 **<RGW USER SECRET ACCESS KEY>** 를 이전 단계의 적절한 디코딩된 데이터로 바꿉니다.

d.

<bucket-name> 을 기존 **RGW** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

e.

<RGW 엔드포인트**>** 를 가져오려면 **RADOS** 개체 게이트웨이 **S3** 끝점 액세스를 참조하십시오.

출력은 다음과 유사합니다.

```
INFO[0001]  Exists: NooBaa "noobaa"
INFO[0002]  Created: BackingStore "rgw-resource"
INFO[0002]  Created: Secret "backing-store-secret-rgw-resource"
```

YAML을 사용하여 백업 저장소를 생성할 수도 있습니다.

1.

CephObjectStore 사용자를 만듭니다. 이렇게 하면 **RGW** 인증 정보가 포함된 보안도 생성됩니다.

```
apiVersion: ceph.rook.io/v1
kind: CephObjectStoreUser
metadata:
  name: <RGW-Username>
  namespace: openshift-storage
spec:
  store: ocs-storagecluster-cephobjectstore
  displayName: "<Display-name>"
```

a.

<RGW-Username> 및 **<Display-name>** 을 고유한 사용자 이름 및 표시 이름으로 바꿉니다.

2.

S3 호환 백업 저장소에 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
  - noobaa.io/finalizer
```

```
  labels:
    app: noobaa
  name: <backingstore-name>
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <RGW endpoint>
    secret:
      name: <backingstore-secret-name>
      namespace: openshift-storage
    signatureVersion: v4
    targetBucket: <RGW-bucket-name>
  type: s3-compatible
```

a. **<backingstore-secret-name>** 을 이전 단계에서 **CephObjectStore** 로 생성된 시크릿 이름으로 바꿉니다.

b. **<bucket-name>** 을 기존 **RGW** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

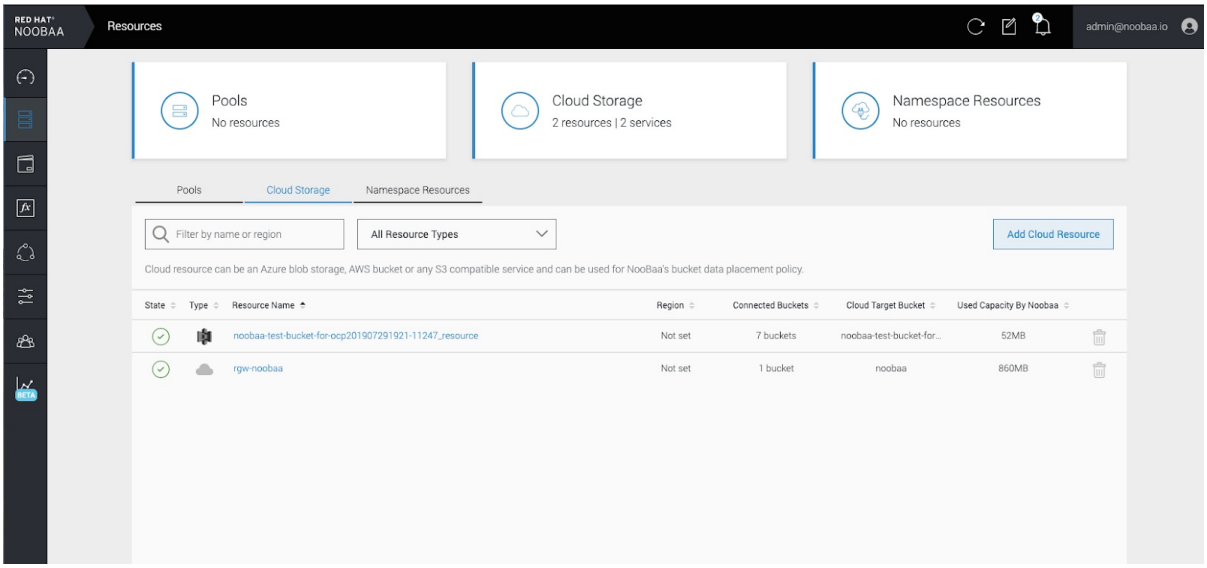
c. **<RGW** 엔드포인트**>** 를 가져오려면 **RADOS** 개체 게이트웨이 **S3** 끝점 액세스를 참조하십시오.

4.4. 사용자 인터페이스를 사용하여 하이브리드 및 MULTICLOUD용 스토리지 리소스 추가

절차

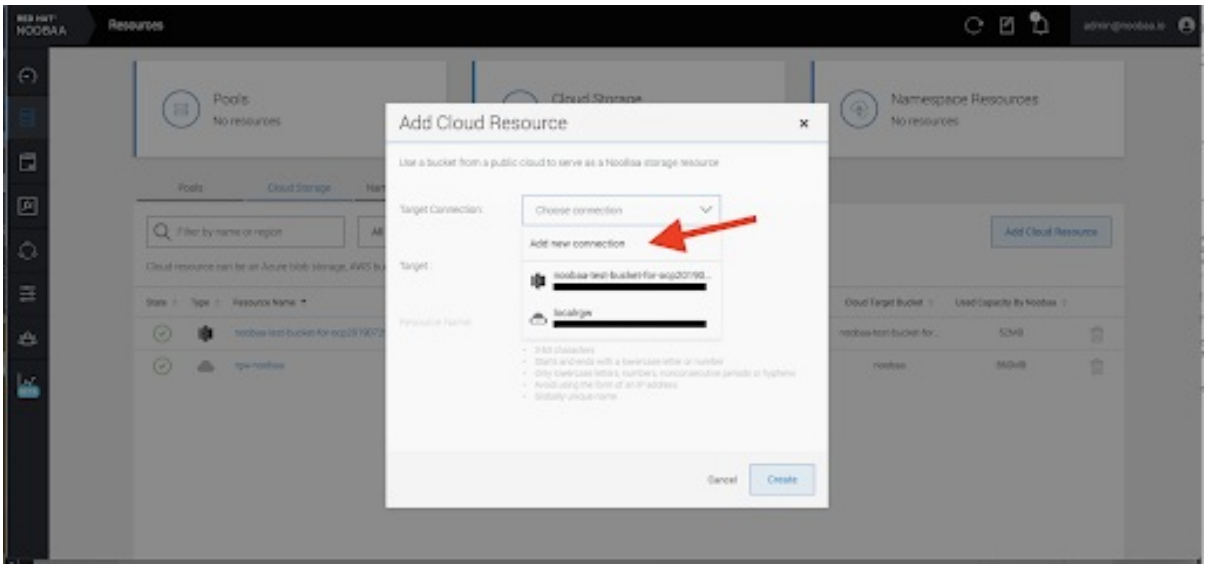
1. **OpenShift Storage** 콘솔에서 스토리지 → 개요 → 오브젝트 탭 → **Multicloud Object Gateway** 링크를 클릭합니다.

2. 아래에 강조 표시된 왼쪽에서 **Resources(**리소스 **)** 탭을 선택합니다. 채워지는 목록에서 클라우드 리소스 추가를 선택합니다.



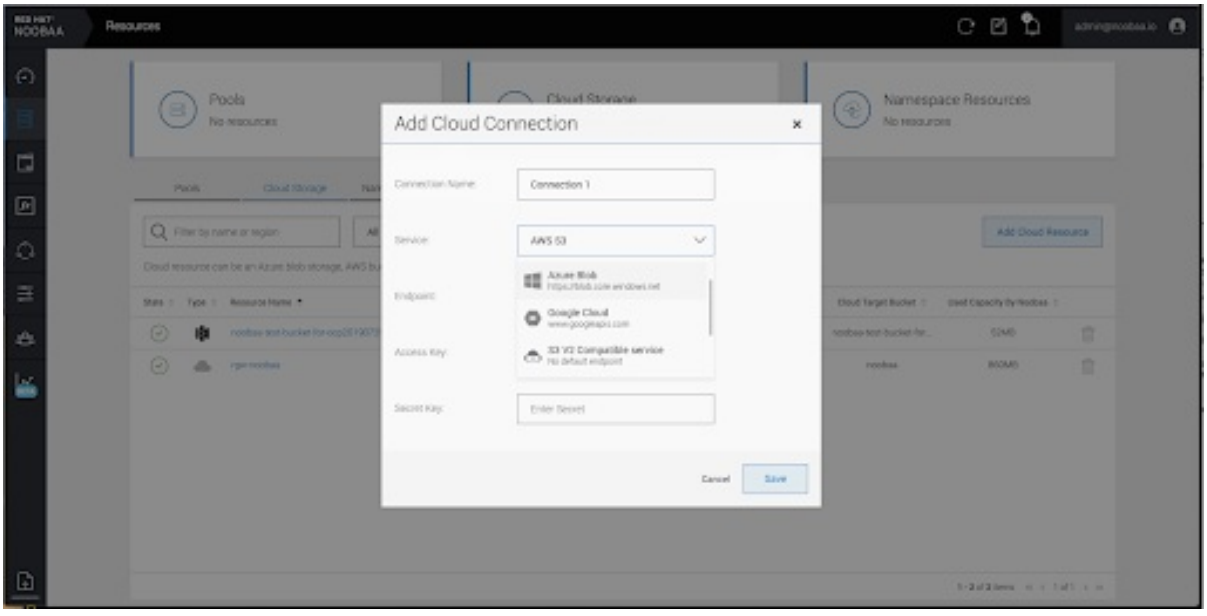
3.

Add new connection(새 연결 추가)을 선택합니다.

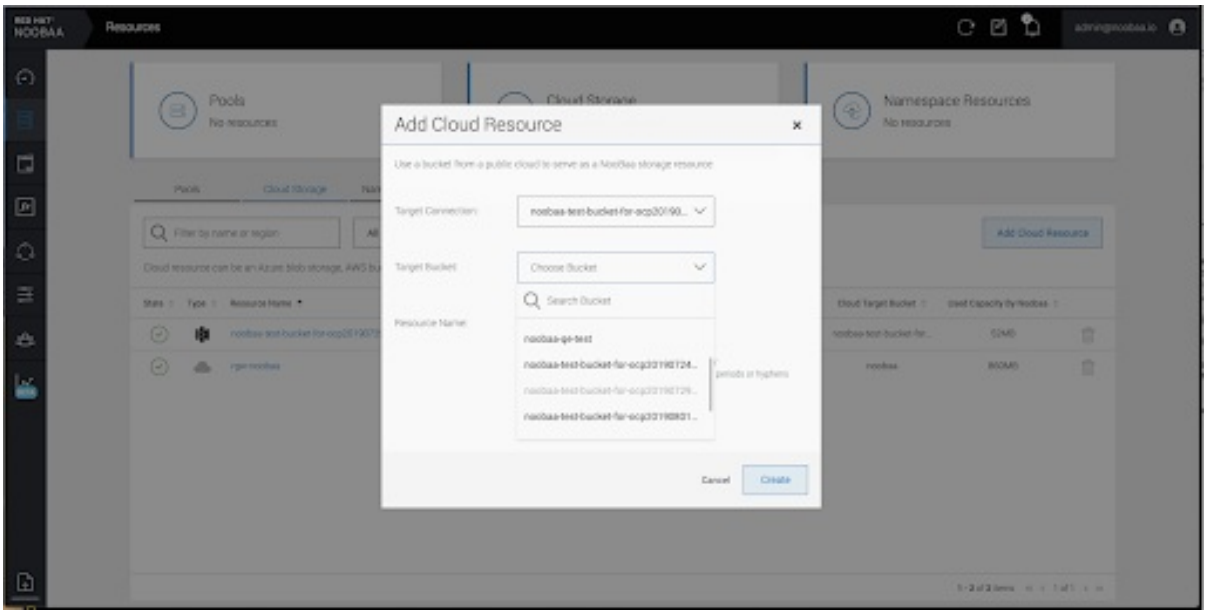


4.

관련 기본 클라우드 공급자 또는 S3 호환 옵션을 선택하고 세부 정보를 입력합니다.



5. 새로 생성된 연결을 선택하고 기존 버킷에 매핑합니다.



6. 이 단계를 반복하여 필요한 만큼의 백업 저장소를 만듭니다.

참고

NooBaa UI에서 생성된 리소스는 OpenShift UI 또는 MCG CLI에서 사용할 수 없습니다.

4.5. 새 버킷 클래스 생성

버킷 클래스는 OBC(오브젝트 버킷 클래스)에 대한 계층화 정책 및 데이터 배치를 정의하는 버킷 클래스를 나타내는 CRD입니다.

OpenShift Container Storage에서 버킷 클래스를 생성하려면 다음 절차를 사용하십시오.

절차

1. OpenShift 웹 콘솔의 왼쪽 창에서 Operators → 설치된 Operator 를 클릭하여 설치된 Operator를 확인합니다.

2. OpenShift Container Storage Operator를 클릭합니다.

3.

OpenShift Container Storage Operator 페이지에서 오른쪽 스크롤하고 버킷 클래스 탭을 클릭합니다.

4. **Create** 버킷 클래스(버킷 클래스 만들기)를 클릭합니다.

5. 새 버킷 클래스 생성 페이지에서 다음을 수행합니다.

   a. 버킷 클래스 유형을 선택하고 버킷 클래스 이름을 입력합니다.

      i. 버킷 클래스 유형을 선택합니다. 다음 옵션 중 하나를 선택합니다.

      - 네임스페이스

        데이터는 중복 제거, 압축 또는 암호화를 수행하지 않고 **NamespaceStores** 에 저장됩니다.

      - **Standard**

        데이터는 **MCG(Multicloud Object Gateway)**에서 사용하며 중복, 압축 및 암호화됩니다.

        기본적으로 **Standard** 가 선택됩니다.

      ii. 버킷 클래스 이름을 입력합니다.

      iii. 다음을 클릭합니다.

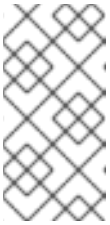
   b. **Placement Policy**(배치 정책)에서 **Tier 1 - Policy Type**(계층 **1 - Policy Type**)을 선택하고 **Next** (다음)를 클릭합니다. 요구 사항에 따라 옵션 중 하나를 선택할 수 있습니다.

      - 분산 을 사용하면 선택한 리소스 전체에 데이터를 분산할 수 있습니다.

- 미러 를 사용하면 선택한 리소스에서 데이터를 완전히 복제할 수 있습니다.

- **Add Tier** (계층 추가)를 클릭하여 다른 정책 계층을 추가합니다.

c. **Tier 1 - Policy Type**을 **Spread**로 선택한 경우 사용 가능한 목록에서 **atleast** 하나의 백업 저장소 리소스를 선택하고 **Next** (다음)를 클릭합니다. 또는 새 백업 저장소를 만들 수도 있습니다.



참고

이전 단계에서 **Policy Type**(정책 유형)을 **mirror**로 선택할 때 **atleast 2** 백업 저장소를 선택해야 합니다.

d. 버킷 클래스 설정을 검토하고 확인합니다.

e. **Create** 버킷 클래스(버킷 클래스 만들기)를 클릭합니다.

검증 단계

1. **Operators** → 설치된 **Operators**를 클릭합니다.

2. **OpenShift Container Storage Operator**를 클릭합니다.

3. 새 버킷 클래스를 검색하거나 버킷 클래스 탭을 클릭하여 모든 버킷 클래스를 확인합니다.

4.6. 버킷 클래스 편집

Openshift 웹 콘솔에서 편집 버튼을 클릭하여 **YAML** 파일을 통해 버킷 클래스 구성 요소를 편집하려면 다음 절차를 사용합니다.

사전 요구 사항

- **OpenShift**에 대한 관리자 액세스.

절차

1. OpenShift 웹 콘솔에 로그인합니다.
2. Operators → 설치된 Operators를 클릭합니다.
3. OpenShift Container Storage Operator를 클릭합니다.
4. OpenShift Container Storage Operator 페이지에서 오른쪽 스크롤하고 버킷 클래스 탭을 클릭합니다.
5. 편집할 버킷 클래스 옆에 있는 작업 메뉴(kube)를 클릭합니다.
6. Edit 버킷 Class(버킷 클래스 편집)를 클릭합니다.
7. YAML 파일로 리디렉션되어 이 파일에 필요한 변경 작업을 수행하고 저장을 클릭합니다.

4.7. 버킷 클래스의 백업 저장소 편집

다음 절차에 따라 기존 Multicloud Object Gateway 버킷 클래스를 편집하여 버킷 클래스에 사용된 기본 백업 저장소를 변경합니다.

사전 요구 사항

- OpenShift 웹 콘솔에 대한 관리자 액세스.
- 버킷 클래스.
- 백업 저장소.

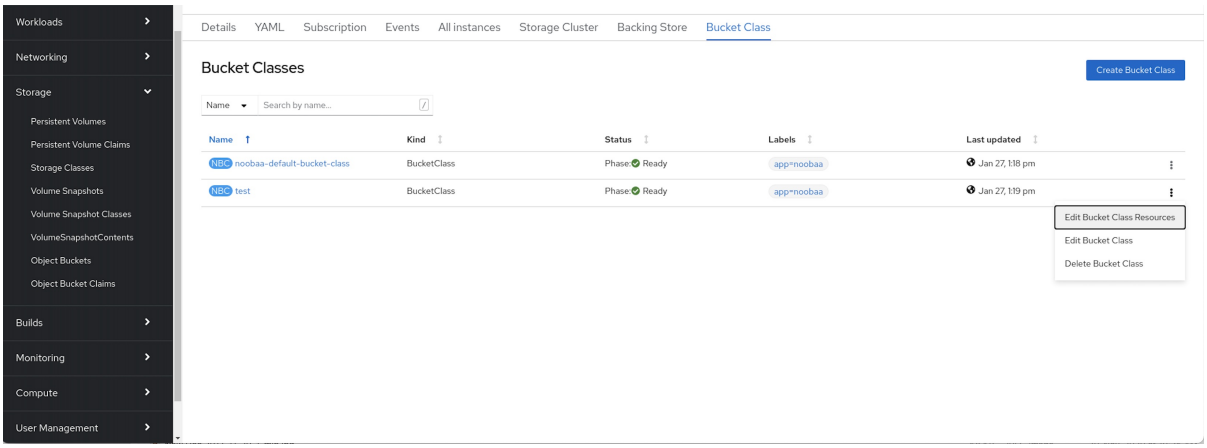
절차

1. **Operators** → 설치된 **Operator**를 클릭하여 설치된 **Operator**를 확인합니다.

2. **OpenShift Container Storage Operator**를 클릭합니다.

3. 버킷 클래스 탭을 클릭합니다.

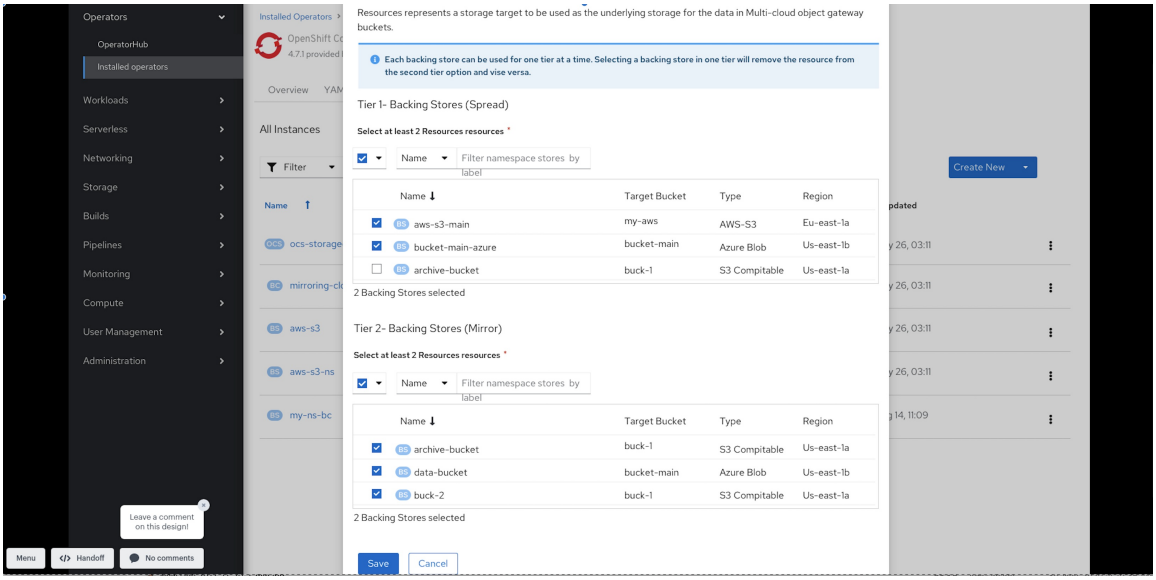
4. 편집할 버킷 클래스 옆에 있는 작업 메뉴(**kube**)를 클릭합니다.



5. **Edit** 버킷 **Class Resources(** 버킷 클래스 리소스 편집)를 클릭합니다.

6. 버킷 클래스 리소스 편집 페이지에서 버킷 클래스에 백업 저장소를 추가하거나 버킷 클래스에서 백업 저장소를 제거하여 버킷 클래스 리소스를 편집합니다. 하나 또는 두 개의 계층과 다양한 배치 정책을 사용하여 생성된 버킷 클래스 리소스를 편집할 수도 있습니다.

   - 버킷 클래스에 백업 저장소를 추가하려면 백업 저장소의 이름을 선택합니다.
   - 버킷 클래스에서 백업 저장소를 제거하려면 백업 저장소의 이름을 지웁니다.


Operators
OperatorHub
Installed operators
Workloads
Serverless
Networking
Storage
Builds
Pipelines
Monitoring
Compute
User Management
Administration
Installed Operators
Overview
All Instances
Filter
Name
Create New
Resources represents a storage target to be used as the underlying storage for the data in Multi-cloud object gateway buckets.
Each backing store can be used for one tier at a time. Selecting a backing store in one tier will remove the resource from the second tier option and vise versa.
Tier 1- Backing Stores (Spread)
Select at least 2 Resources resources *
Filter namespace stores by label
Name
Target Bucket
Type
Region
aws-s3-main
my-aws
AWS-S3
Eu-east-1a
bucket-main-azure
bucket-main
Azure Blob
Us-east-1b
archive-bucket
buck-1
S3 Compitable
Us-east-1a
2 Backing Stores selected
Tier 2- Backing Stores (Mirror)
Select at least 2 Resources resources *
data-bucket
buck-2
2 Backing Stores selected
Save
Cancel
Leave a comment on this design!
Menu
Handoff
No comments

7. **저장을 클릭합니다.**

5장. 네임스페이스 버킷 관리

네임 스페이스 버킷을 사용하면 여러 공급자의 데이터 리포지토리를 함께 연결할 수 있으므로 단일 통합 뷰를 통해 모든 데이터와 상호 작용할 수 있습니다. 각 공급자와 연결된 오브젝트 버킷을 네임 스페이스 버킷에 추가하고 네임 스페이스 버킷을 통해 데이터에 액세스하여 모든 오브젝트 버킷을 한 번에 확인합니다. 이를 통해 원하는 스토리지 공급자에 쓰는 동시에 다른 여러 스토리지 공급자로부터 읽을 수 있어 새 스토리지 공급자로 마이그레이션하는 데 드는 비용을 크게 줄일 수 있습니다.

참고

네임 스페이스 버킷은 쓰기 대상이 사용 가능하고 작동하는 경우에만 사용할 수 있습니다.

5.1. 네임스페이스 버킷의 오브젝트에 대한 AMAZON S3 API 끝점

Amazon S3(Simple Storage Service) API를 사용하여 네임스페이스 버킷의 오브젝트와 상호 작용할 수 있습니다.

Red Hat OpenShift Container Storage 4.6 이후에서는 다음과 같은 네임 스페이스 버킷 작업을 지원합니다.

- **ListObjectVersions**
- **ListObjects**
- **PutObject**
- **CopyObject**
- **ListParts**
- **CreateMultipartUpload**

- CompleteMultipartUpload
- UploadPart
- UploadPartCopy
- AbortMultipartUpload
- GetObjectAcl
- GetObject
- HeadObject
- DeleteObject
- DeleteObjects

이러한 작업 및 사용 방법에 대한 최신 정보는 **Amazon S3 API** 참조 문서를 참조하십시오.

추가 리소스

- Amazon S3 REST API 참조
- Amazon S3 CLI 참조

5.2. MULTICLOUD OBJECT GATEWAY CLI 및 YAML을 사용하여 네임스페이스 버킷 추가

네임스페이스 버킷에 대한 자세한 내용은 네임스페이스 버킷 관리를 참조하십시오.

배포 유형 및 YAML 또는 Multicloud Object Gateway CLI를 사용할지 여부에 따라 다음 절차 중 하나를 선택하여 네임 스페이스 버킷을 추가합니다.

- YAML을 사용하여 AWS S3 네임 스페이스 버킷 추가
- YAML을 사용하여 IBM COS 네임 스페이스 버킷 추가
- Multicloud Object Gateway CLI를 사용하여 AWS S3 네임 스페이스 버킷 추가
- Multicloud Object Gateway CLI를 사용하여 IBM COS 네임 스페이스 버킷 추가

5.2.1. YAML을 사용하여 AWS S3 네임 스페이스 버킷 추가

사전 요구 사항

- 실행 중인 OpenShift Container Storage Platform
- Multicloud Object Gateway에 액세스하고 2장, 애플리케이션을 사용하여 Multicloud Object Gateway에액세스

절차

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
name: <namespacestore-secret-name>
type: Opaque
data:
AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
```

a. Base64를 사용하여 자체 AWS 액세스 키 ID 및 시크릿 액세스 키를 제공하고 인코딩해야 하며 <AWS ACCESS KEY ID ENCODED in BASE64> 및 < AWS SECRET ACCESS KEY ENCODED in BASE64> 대신 결과를 사용해야 합니다. ii. <namespacestore-secret-name> 을 고유한 이름으로 바꿉니다.

2. **OpenShift CRD(Custom Resource Definitions)**를 사용하여 네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object Gateway** 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. 네임스페이스 저장소 리소스를 생성하려면 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <resource-name>
  namespace: openshift-storage
spec:
  awsS3:
    secret:
      name: <namespacestore-secret-name>
      namespace: <namespace-secret>
    targetBucket: <target-bucket>
  type: aws-s3
```

a. **<resource-name>** 을 리소스에 제공할 이름으로 바꿉니다.

b. **<namespacestore-secret-name>** 을 **1** 단계에서 생성된 보안으로 교체합니다.

c. **<namespace-secret>** 을 보안을 찾을 수 있는 네임스페이스로 바꿉니다.

d. **<target-bucket>** 을 네임 스페이스 저장소에 대해 생성한 대상 버킷으로 바꿉니다.

3. 네임스페이스 버킷에 대한 네임스페이스 정책을 정의하는 네임스페이스 버킷 클래스를 생성합니다. 네임스페이스 정책에는 단일 또는 **multi** 의 유형이 필요합니다.

- **type single** 의 네임스페이스 정책에는 다음과 같은 구성이 필요합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
```

```
spec:
  namespacePolicy:
    type:
    single:
      resource: <resource>
```

<my-bucket-class> 를 고유한 네임 스페이스 버킷 클래스 이름으로 바꿉니다.

<resource> 를 네임 스페이스 버킷의 읽기 및 쓰기 대상을 정의할 단일 네임 스페이스 저장소의 이름으로 교체합니다.

- **type multi** 의 네임스페이스 정책에는 다음과 같은 구성이 필요합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type: Multi
    multi:
      writeResource: <write-resource>
      readResources:
      - <read-resources>
      - <read-resources>
```

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<write-resource> 를 단일 네임 스페이스 저장소 이름으로 교체하여 네임 스페이스 버킷의 쓰기 대상을 정의합니다.

<read-resources> 를 네임 스페이스 버킷의 읽기 대상을 정의할 **namespace-stores**의 이름 목록으로 바꿉니다.

4. 다음 **YAML**을 적용하여 **2**단계에 정의된 버킷 클래스를 사용하는 **OBC(**개체 버킷 클래스**)** 리소스를 사용하여 버킷을 생성합니다.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
```

```
metadata:
  name: <resource-name>
  namespace: openshift-storage
spec:
  generateBucketName: <my-bucket>
  storageClassName: noobaa.noobaa.io
  additionalConfig:
    bucketclass: <my-bucket-class>
```

참고

IBM Power Systems 및 **IBM Z** 인프라의 경우 **openshift-storage.noobaa.io**로 **use storageClassName**

a. **<my-bucket-class>** 를 이전 단계에서 생성한 버킷 클래스로 바꿉니다.

Operator가 **OBC**를 프로비저닝하면 **Multicloud Object Gateway**에서 버킷이 생성되고 **Operator**는 **OBC**의 동일한 네임스페이스에서 **OBC**의 동일한 이름으로 **Secret** 및 **ConfigMap**을 생성합니다.

5.2.2. YAML을 사용하여 IBM COS 네임 스페이스 버킷 추가

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform**
- **Multicloud Object Gateway**에 액세스하고 **2**장, 애플리케이션을 사용하여 **Multicloud Object Gateway**에액세스

절차

1. 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
name: <namespacestore-secret-name>
type: Opaque
data:
IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED IN BASE64>
```

a. **Base64**를 사용하여 자체 **IBM COS** 액세스 키 **ID** 및 비밀 액세스 키를 제공하고 인코딩해야 하며 **<IBM COS COS** 키 **ID ENCODED IN BASE64>** 및 **'< IBM COS SECRET ACCESS KEY ENCODED IN BASE64>** 대신 결과를 사용해야 합니다.

b. **<namespacestore-secret-name>** 을 고유한 이름으로 바꿉니다.

2. **OpenShift CRD(Custom Resource Definitions)**를 사용하여 네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object Gateway** 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. 네임스페이스 저장소 리소스를 생성하려면 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: bs
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <IBM COS ENDPOINT>
    secret:
      name: <namespacestore-secret-name>
      namespace: <namespace-secret>
    signatureVersion: v2
    targetBucket: <target-bucket>
  type: ibm-cos
```

a. **<IBM COS ENDPOINT>** 를 적절한 **IBM COS** 엔드포인트로 바꿉니다.

b. **<namespacestore-secret-name>** 을 **1** 단계에서 생성된 보안으로 교체합니다.

c. **<namespace-secret>** 을 보안을 찾을 수 있는 네임스페이스로 바꿉니다.

d. **<target-bucket>** 을 네임 스페이스 저장소에 대해 생성한 대상 버킷으로 바꿉니다.

3. 네임스페이스 버킷에 대한 네임스페이스 정책을 정의하는 네임스페이스 버킷 클래스를 생성합니다. 네임스페이스 정책에는 단일 또는 **multi** 의 유형이 필요합니다.

- **type single** 의 네임스페이스 정책에는 다음과 같은 구성이 필요합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type:
    single:
      resource: <resource>
```

<my-bucket-class> 를 고유한 네임 스페이스 버킷 클래스 이름으로 바꿉니다.

<resource> 를 단일 네임 스페이스 저장소 이름으로 교체하여 네임 스페이스 버킷의 읽기 및 쓰기 대상을 정의합니다.

- **type multi** 의 네임스페이스 정책에는 다음과 같은 구성이 필요합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <my-bucket-class>
  namespace: openshift-storage
spec:
  namespacePolicy:
    type: Multi
    multi:
      writeResource: <write-resource>
      readResources:
      - <read-resources>
      - <read-resources>
```

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<write-resource> 를 단일 네임 스페이스 저장소 이름으로 교체하여 네임 스페이스 버킷의 쓰기 대상을 정의합니다.

<read-resources> 를 네임 스페이스 버킷의 읽기 대상을 정의하는 **namespace-store**의 이름 목록으로 바꿉니다.

4. 다음 **YAML**을 적용하여 **2**단계에 정의된 버킷 클래스를 사용하는 **OBC(**개체 버킷 클래스**)** 리소스를 사용하여 버킷을 생성합니다.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <resource-name>
  namespace: openshift-storage
spec:
  generateBucketName: <my-bucket>
  storageClassName: noobaa.noobaa.io
  additionalConfig:
    bucketclass: <my-bucket-class>
```

참고

IBM Power Systems 및 **IBM Z** 인프라의 경우 **openshift-storage.noobaa.io**로 **use storageClassName**

a. **<my-bucket-class>** 를 이전 단계에서 생성한 버킷 클래스로 바꿉니다.

Operator가 **OBC**를 프로비저닝하면 **Multicloud Object Gateway**에서 버킷이 생성되고 **Operator**는 **OBC**의 동일한 네임스페이스에서 **OBC**의 동일한 이름으로 **Secret** 및 **ConfigMap**을 생성합니다.

5.2.3. Multicloud Object Gateway CLI를 사용하여 AWS S3 네임 스페이스 버킷 추가

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform**
- **Multicloud Object Gateway**에 액세스하고 **2**장, 애플리케이션을 사용하여 **Multicloud Object Gateway**에액세스
- **Multicloud Object Gateway** 명령줄 인터페이스를 다운로드합니다.

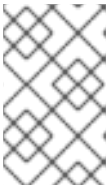
```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package** 에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.



참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. 네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object Gateway** 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa namespacestore create aws-s3 <namespacestore > --access-key <AWS ACCESS KEY> --secret-key <AWS SECRET ACCESS KEY> --target-bucket <bucket-name> -n openshift-storage
```

   a. **<namespacestore>** 를 네임 스페이스 저장소의 이름으로 바꿉니다.

   b. **<AWS ACCESS KEY>** 및 **<AWS SECRET ACCESS KEY>** 를 이 목적을 위해 생성한 **AWS** 액세스 키 **ID** 및 시크릿 액세스 키로 바꿉니다.

   c. **<bucket-name>** 을 기존 **AWS** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

2. 네임스페이스 버킷에 대한 네임스페이스 정책을 정의하는 네임스페이스 버킷 클래스를 생성

합니다. 네임스페이스 정책에는 단일 또는 **multi** 의 유형이 필요합니다.

- 다음 명령을 실행하여 단일 유형의 네임스페이스 정책으로 네임 스페이스 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass single <my-bucket-class> --resource <resource> -n openshift-storage
```

<resource-name> 을 리소스를 제공할 이름으로 바꿉니다.

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<resource> 를 네임 스페이스 버킷의 읽기 및 쓰기 대상을 정의하는 단일 네임 스페이스 저장소로 바꿉니다.

- 다음 명령을 실행하여 유형 **multi** 의 네임스페이스 정책으로 네임 스페이스 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass multi <my-bucket-class> --write-resource <write-resource> --read-resources <read-resources> -n openshift-storage
```

<resource-name> 을 리소스를 제공할 이름으로 바꿉니다.

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<write-resource> 를 네임 스페이스 버킷의 쓰기 대상을 정의할 단일 네임 스페이스 저장소로 바꿉니다.

<read-resources> 를 네임 스페이스 버킷의 읽기 대상을 정의하는 쉼표로 구분된 네임 스페이스 저장소 목록으로 바꿉니다.

3. 다음 명령을 실행하여 **2**단계에 정의된 버킷 클래스를 사용하는 **OBC(**개체 버킷 클래스**)** 리소스를 사용하여 버킷을 생성합니다.

```
noobaa obc create my-bucket-claim -n openshift-storage --app-namespace my-app --
bucketclass <custom-bucket-class>
```

a. **<bucket-name>** 을 선택한 버킷 이름으로 바꿉니다.

b. **<custom-bucket-class>** 를 2단계에서 생성된 버킷 클래스의 이름으로 바꿉니다.

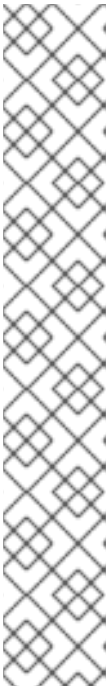
Operator가 **OBC**를 프로비저닝하면 **Multicloud Object Gateway**에서 버킷이 생성되고 **Operator**는 **OBC**의 동일한 네임스페이스에서 **OBC**의 동일한 이름으로 **Secret** 및 **ConfigMap**을 생성합니다.

5.2.4. Multicloud Object Gateway CLI를 사용하여 IBM COS 네임 스페이스 버킷 추가

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform**
- **Multicloud Object Gateway**에 액세스하고 **2**장, 애플리케이션을 사용하여 **Multicloud Object Gateway**에액세스
- **Multicloud Object Gateway** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```



참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

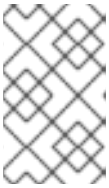
- **IBM Power Systems**의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package** 에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.



참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. 네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object Gateway** 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa namespacestore create ibm-cos <namespacestore> --endpoint <IBM COS ENDPOINT> --access-key <IBM ACCESS KEY> --secret-key <IBM SECRET ACCESS KEY> --target-bucket <bucket-name> -n openshift-storage
```

a. **<namespacestore>** 를 네임 스페이스 저장소의 이름으로 바꿉니다.

b. **<IBM ACCESS KEY>, <IBM SECRET ACCESS KEY>, <IBM COS ENDPOINT>** 를 **IBM** 액세스 키 **ID,** 시크릿 액세스 키 및 기존 **IBM** 버킷 위치에 해당하는 해당 지역 엔드포인트로 바꿉니다.

c.

<bucket-name> 을 기존 **IBM** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

2.

네임스페이스 버킷에 대한 네임스페이스 정책을 정의하는 네임스페이스 버킷 클래스를 생성합니다. 네임스페이스 정책에는 단일 또는 **multi** 의 유형이 필요합니다.

- 다음 명령을 실행하여 단일 유형의 네임스페이스 정책으로 네임 스페이스 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass single <my-bucket-class> --resource
<resource> -n openshift-storage
```

<resource-name> 을 리소스를 제공할 이름으로 바꿉니다.

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<resource> 를 네임 스페이스 버킷의 읽기 및 쓰기 대상을 정의하는 단일 네임 스페이스 저장소로 바꿉니다.

- 다음 명령을 실행하여 유형 **multi** 의 네임스페이스 정책으로 네임 스페이스 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass multi <my-bucket-class> --write-
resource <write-resource> --read-resources <read-resources> -n openshift-storage
```

<resource-name> 을 리소스를 제공할 이름으로 바꿉니다.

<my-bucket-class> 를 고유한 버킷 클래스 이름으로 바꿉니다.

<write-resource> 를 네임 스페이스 버킷의 쓰기 대상을 정의할 단일 네임 스페이스 저장소로 바꿉니다.

<read-resources> 를 네임 스페이스 버킷의 읽기 대상을 정의하는 쉼표로 구분된 네임 스페이스 저장소 목록으로 바꿉니다.

3. 다음 명령을 실행하여 2단계에 정의된 버킷 클래스를 사용하는 OBC(개체 버킷 클래스) 리소스를 사용하여 버킷을 생성합니다.

```
noobaa obc create my-bucket-claim -n openshift-storage --app-namespace my-app --bucketclass <custom-bucket-class>
```

a. **<bucket-name>** 을 선택한 버킷 이름으로 바꿉니다.

b. **<custom-bucket-class>** 를 2단계에서 생성된 버킷 클래스의 이름으로 바꿉니다.

Operator가 OBC를 프로비저닝하면 Multicloud Object Gateway에서 버킷이 생성되고 Operator는 OBC의 동일한 네임스페이스에서 OBC의 동일한 이름으로 Secret 및 ConfigMap을 생성합니다.

5.3. OPENSHIFT CONTAINER PLATFORM 사용자 인터페이스를 사용하여 네임스페이스 버킷 추가

OpenShift Container Storage 4.8 릴리스를 통해 OpenShift Container Platform 사용자 인터페이스를 사용하여 네임스페이스 버킷을 추가할 수 있습니다. 네임스페이스 버킷에 대한 자세한 내용은 네임스페이스 버킷 관리를 참조하십시오.

사전 요구 사항

- OpenShift Container Storage Operator가 설치된 OpenShift Container Platform
- Multicloud Object Gateway에 대한 액세스.

절차

1. OpenShift 웹 콘솔에 로그인합니다.
2. OpenShift 웹 콘솔의 왼쪽 창에서 Operators → 설치된 Operator 를 클릭하여 설치된 Operator를 확인합니다.
3. OpenShift Container Storage Operator를 클릭합니다. 표시되지 않으면 모든 프로젝트 또는 openshift-storage 네임스페이스로 이동합니다.

4. **Namespace Store (네임스페이스 저장소) 탭을 클릭하여 네임스페이스 버킷에서 사용할 네임스페이스 저장소 리소스를 생성합니다.**

5. **네임 스페이스 저장소 페이지에서 다음을 수행합니다.**

   a. **Create namespace store (네임스페이스 저장소 만들기)를 클릭합니다.**

   b. **네임스페이스 저장소 이름을 입력합니다.**

   c. **공급업체 선택.**

   d. **지역 선택.**

   e. **기존 시크릿을 선택하거나 Switch to credentials 를 클릭하여 보안 키 및 시크릿 액세스 키를 입력하여 시크릿을 생성합니다.**

   f. **대상 버킷을 선택합니다.**

   g. **생성을 클릭합니다.**

   h. **네임스페이스 저장소가 Ready 상태인지 확인합니다.**

   i. **원하는 양의 리소스가 있을 때까지 반복합니다.**

6. **버킷 클래스 탭을 클릭합니다.**

7. **Create a newucket Class(새 버킷 클래스 만들기)를 클릭합니다.**

8. **Create a new Bucket Class (새 버킷 클래스 생성) 페이지에서 다음을 수행합니다.**

a. 네임스페이스 라디오 버튼을 선택합니다.

b. 버킷 클래스 이름을 입력합니다.

c. 설명 추가(선택 사항).

d. 다음을 클릭합니다.

9. 다음 페이지에서 네임스페이스 버킷에 대한 네임스페이스 정책 유형을 선택한 다음 **Next** (다음)를 클릭합니다.

10. 대상 리소스를 선택합니다.

- 네임스페이스 정책 유형이 **Single** 인 경우 읽기 리소스를 선택해야 합니다.
- 네임스페이스 정책 유형이 **Multi** 인 경우 읽기 리소스와 쓰기 리소스를 선택해야 합니다.
- 네임스페이스 정책 유형이 **Cache** 인 경우 네임 스페이스 버킷의 읽기 및 쓰기 대상을 정의하는 **Hub** 네임 스페이스 저장소를 선택해야 합니다.

11. 다음을 클릭합니다.

12. 새 버킷 클래스를 검토한 다음 **Create Bucketclass** 를 클릭합니다.

13. **BucketClass** 페이지에서 새로 생성된 리소스가 **Created** 단계에 있는지 확인합니다.

14. **OpenShift** 콘솔에서 스토리지 → 개요 를 클릭하고 오브젝트 탭을 클릭합니다.

15. **Multicloud Object Gateway** (다중 클라우드 개체 게이트웨이)를 클릭합니다.

16. 버킷을 클릭하고 네임 스페이스 버킷 탭을 클릭합니다.

17. 네임스페이스 버킷 생성을 클릭합니다.

    a. **Choose Name** (이름 선택) 탭에서 네임스페이스 버킷 의 이름을 지정하고 **Next** (다음) 를 클릭합니다.

    b. **Set Placement** (배치 설정) 탭에서 다음을 수행합니다.

        i. **Read Policy** (읽기 정책)에서 네임스페이스 버킷이 데이터를 읽도록 **5**단계에서 생성된 각 네임스페이스 리소스의 확인란을 선택합니다.

        ii. 사용 중인 네임스페이스 정책 유형이 **Multi** 이면 **Read Policy**(쓰기 정책)에서 네임스페이스 버킷에서 데이터를 작성해야 하는 네임스페이스 리소스를 지정합니다.

        iii. 다음을 클릭합니다.

    c. 생성을 클릭합니다.

검증

- **State** 열, 예상되는 읽기 리소스 수 및 예상 쓰기 리소스 이름에 녹색 확인 표시가 포함된 네임스페이스 버킷이 나열되는지 확인합니다.

6장. 하이브리드 및 멀티 클라우드 버킷의 데이터 미러링

MCG(Multicloud Object Gateway)는 클라우드 공급자와 클러스터 전체에서 데이터를 포괄하는 프로세스를 간소화합니다.

사전 요구 사항

- 먼저 **MCG**에서 사용할 수 있는 백업 스토리지를 추가해야 하며 4장. *하이브리드 또는 Multicloud 용 스토리지 리소스 추가* 참조하십시오.

그런 다음 데이터 관리 정책 미러링을 반영하는 버킷 클래스를 생성합니다.

절차

미러링 데이터를 설정하는 방법은 다음과 같습니다.

- 6.1절. “MCG 명령줄 인터페이스를 사용하여 데이터를 미러링하는 버킷 클래스 생성”
- 6.2절. “YAML을 사용하여 데이터를 미러링하도록 버킷 클래스 생성”
- 6.3절. “사용자 인터페이스를 사용하여 데이터를 미러링하도록 버킷 구성”

6.1. MCG 명령줄 인터페이스를 사용하여 데이터를 미러링하는 버킷 클래스 생성

1. **MCG** 명령줄 인터페이스에서 다음 명령을 실행하여 미러링 정책으로 버킷 클래스를 생성합니다.

```
$ noobaa bucketclass create placement-bucketclass mirror-to-aws --backingstores=azure-resource,aws-resource --placement Mirror
```

2. 새로 생성된 버킷 클래스를 새 버킷 클레임으로 설정하고 두 위치 간에 미러링될 새 버킷을 생성합니다.

```
$ noobaa obc create  mirrored-bucket --bucketclass=mirror-to-aws
```

6.2. YAML을 사용하여 데이터를 미러링하도록 버킷 클래스 생성

1. 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: <bucket-class-name>
  namespace: openshift-storage
spec:
  placementPolicy:
    tiers:
    - backingStores:
      - <backing-store-1>
      - <backing-store-2>
      placement: Mirror
```

2. 표준 **OBC(**오브젝트 버킷 클레임**)**에 다음 행을 추가합니다.

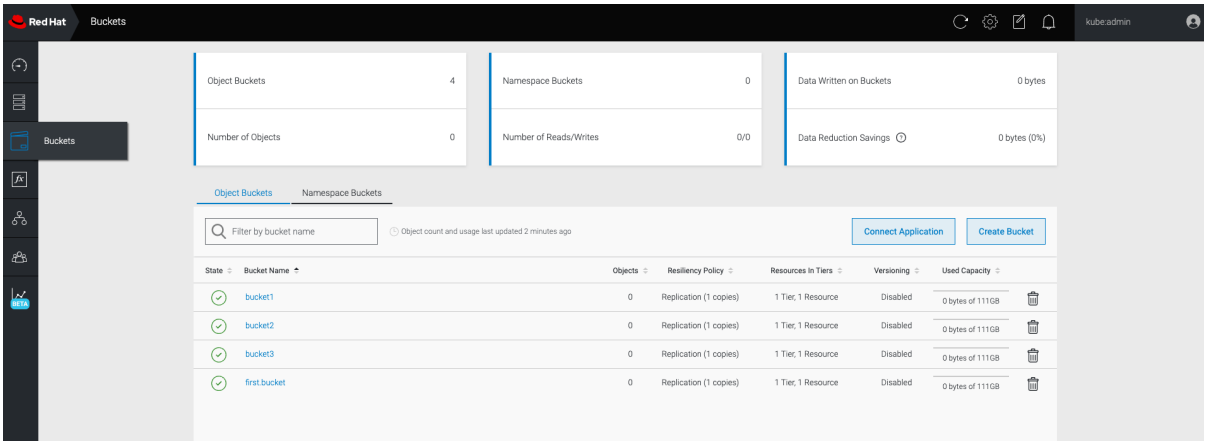
```
additionalConfig:
  bucketclass: mirror-to-aws
```

OBC에 대한 자세한 내용은 *8장. 개체 버킷 클레임* 의 내용을 참조하십시오.

6.3. 사용자 인터페이스를 사용하여 데이터를 미러링하도록 버킷 구성

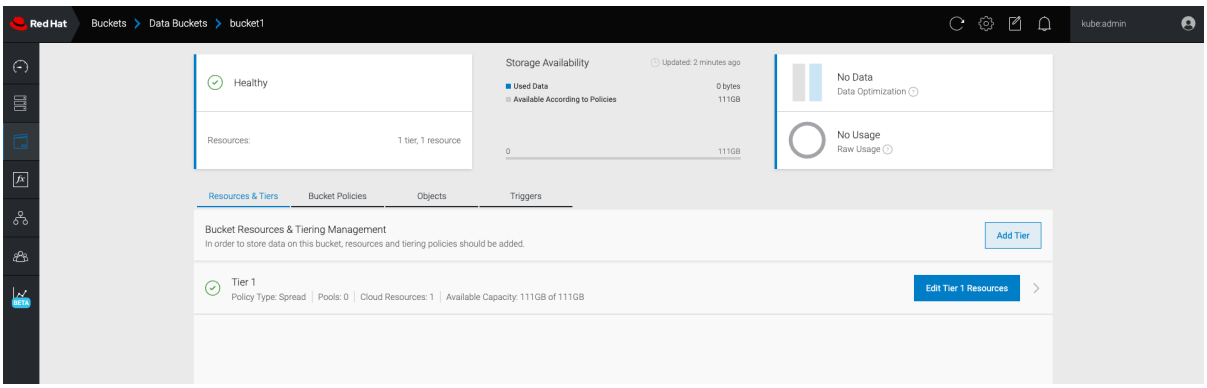
1. **OpenShift Storage** 콘솔에서 스토리지 → 개요 → 오브젝트 탭 → **Multicloud Object Gateway** 링크를 클릭합니다.

2. **NooBaa** 페이지의 왼쪽에 있는 버킷 아이콘을 클릭합니다. 버킷 목록이 표시됩니다.

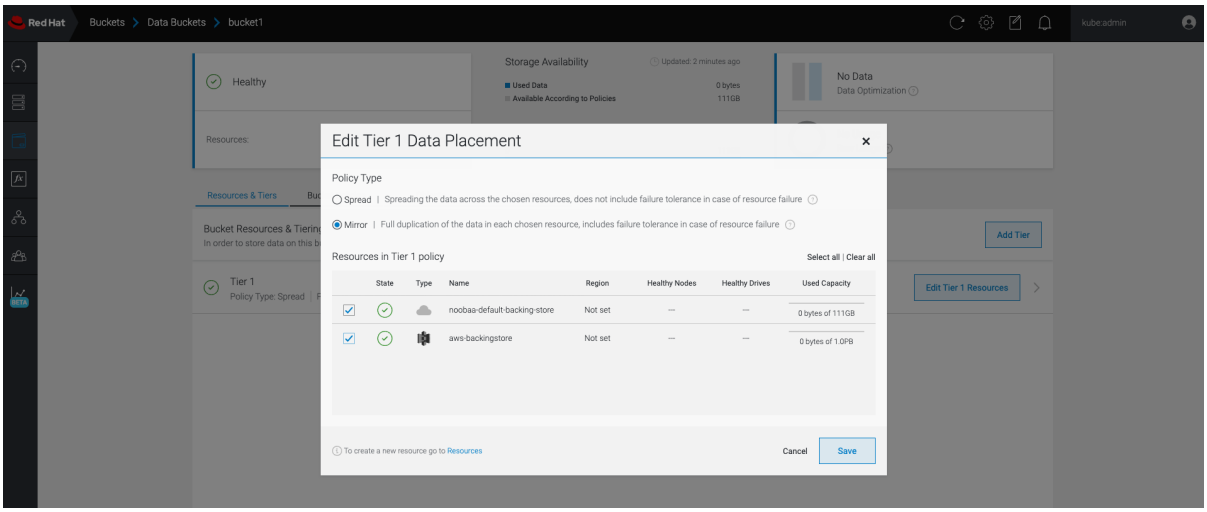


3. 업데이트할 버킷을 클릭합니다.

4. 계층 1 리소스 편집을 클릭합니다.



5. **Mirror** 를 선택하고 이 버킷에 사용할 관련 리소스를 확인합니다. 다음 예제에서 **AWS**에 있는 **noobaa-default-backing-store** 와 **AWS**에 있는 **AWS-backingstore** 간의 데이터가 미러링됩니다.



6. 저장을 클릭합니다.

참고

NooBaa UI에서 생성된 리소스는 **OpenShift UI** 또는 **MCG CLI**에서 사용할 수 없습니다.

7장. MULTICLOUD OBJECT GATEWAY의 버킷 정책

OpenShift Container Storage는 AWS S3 버킷 정책을 지원합니다. 버킷 정책을 사용하면 버킷 및 해당 객체에 대한 액세스 권한을 사용자에게 부여할 수 있습니다.

7.1. 버킷 정책 정보

버킷 정책은 AWS S3 버킷 및 오브젝트에 대한 권한을 부여할 수 있는 액세스 정책 옵션입니다. 버킷 정책은 JSON 기반 액세스 정책 언어를 사용합니다. 액세스 정책 언어에 대한 자세한 내용은 AWS Access Policy Language Overview 를 참조하십시오.

7.2. 버킷 정책 사용

사전 요구 사항

- 실행 중인 OpenShift Container Storage Platform
- Multicloud Object Gateway에 대한 액세스, 참조 *2장. 애플리케이션을 사용하여 Multicloud Object Gateway에 액세스*

절차

Multicloud Object Gateway에서 버킷 정책을 사용하려면 다음을 수행합니다.

1. JSON 형식으로 버킷 정책을 생성합니다. 다음 예제를 참조하십시오.

```
{
    "Version": "NewVersion",
    "Statement": [
        {
            "Sid": "Example",
            "Effect": "Allow",
            "Principal": [
                    "john.doe@example.com"
            ],
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::john_bucket"
            ]
```

```
        }
    ]
}
```

버킷 정책에는 액세스 권한과 관련하여 사용 가능한 많은 요소가 있습니다.

이러한 요소 및 액세스 권한을 제어하는 데 사용할 수 있는 방법에 대한 예제에 대한 자세한 내용은 **AWS Access Policy Language 개요** 를 참조하십시오.

버킷 정책의 예를 보려면 **AWS 버킷 정책 예제** 를 참조하십시오.

S3 사용자를 생성하는 방법은 **7.3절. "Multicloud Object Gateway에서 AWS S3 사용자 생성"** 에서 확인할 수 있습니다.

2. **AWS S3** 클라이언트를 사용하여 **put-bucket-policy** 명령을 사용하여 버킷 정책을 **S3** 버킷에 적용합니다.

```
# aws --endpoint ENDPOINT --no-verify-ssl s3api put-bucket-policy --bucket MyBucket --policy BucketPolicy
```

S3 엔드 포인트로 교체

MyBucket 을 버킷으로 교체하여 정책을 설정합니다.

BucketPolicy 를 버킷 정책 **JSON** 파일로 교체

자체 서명된 기본 인증서를 사용하는 경우 **--no-verify-ssl** 을 추가합니다.

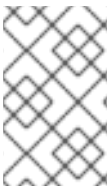
예를 들면 다음과 같습니다.

```
# aws --endpoint https://s3-openshift-storage.apps.gogo44.noobaa.org --no-verify-ssl s3api put-bucket-policy -bucket MyBucket --policy file://BucketPolicy
```

put-bucket-policy 명령에 대한 자세한 내용은 **put-bucket-policy** 에 대한 **AWS CLI** 명령 참조를 참조하십시오.

참고

principal 요소는 버킷과 같은 리소스에 대한 액세스가 허용되거나 거부된 사용자를 지정합니다. 현재는 **NooBaa** 계정만 주체로 사용할 수 있습니다. 개체 버킷 클레임의 경우 **NooBaa**는 자동으로 계정 **obc-account.<generated** 버킷 이름>@noobaa.io 를 생성합니다.



참고

버킷 정책 조건이 지원되지 않습니다.

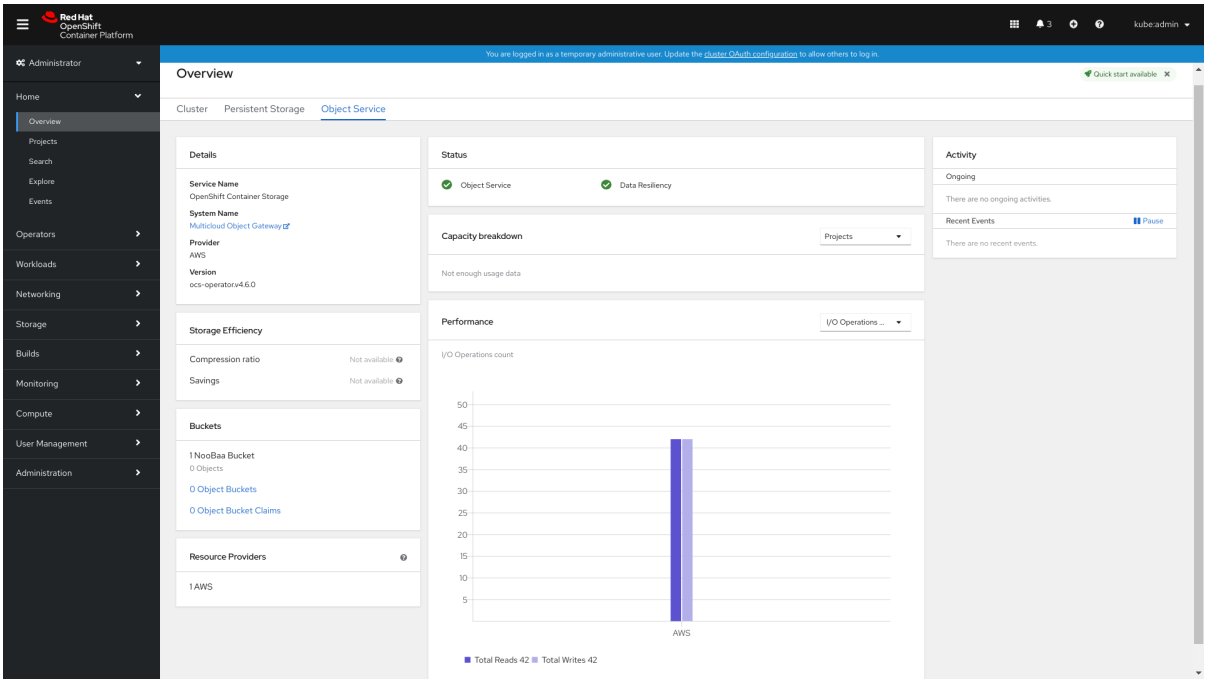
7.3. MULTICLOUD OBJECT GATEWAY에서 AWS S3 사용자 생성

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform**
- **Multicloud Object Gateway**에 대한 액세스, 참조 *2장. 애플리케이션을 사용하여 Multicloud Object Gateway에 액세스*

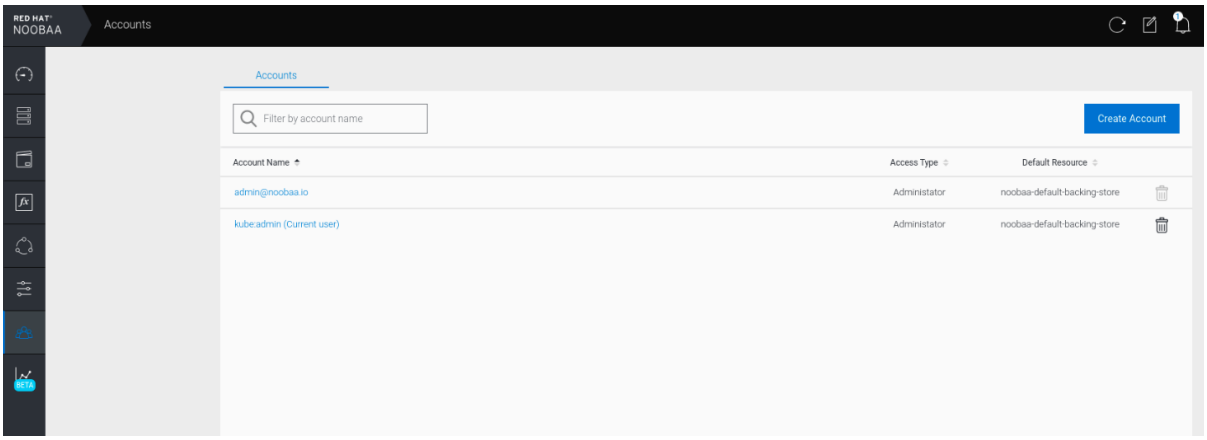
절차

1. **OpenShift Storage** 콘솔에서 스토리지 → 개요 → 오브젝트 탭 → **Multicloud Object Gateway** 링크를 선택합니다.



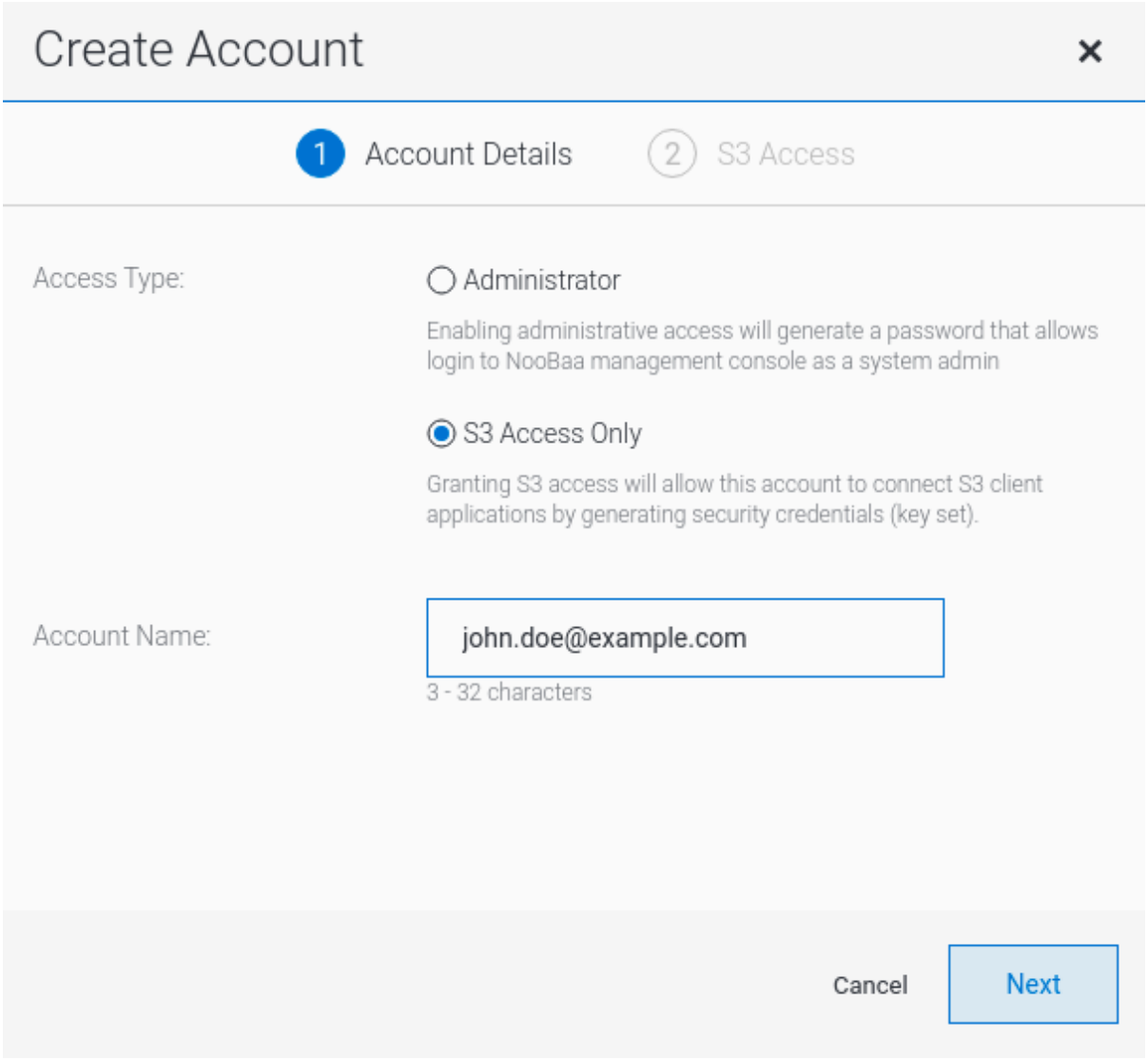
2.

Accounts (계정) 탭에서 Create Account(계정 만들기)를 클릭합니다.

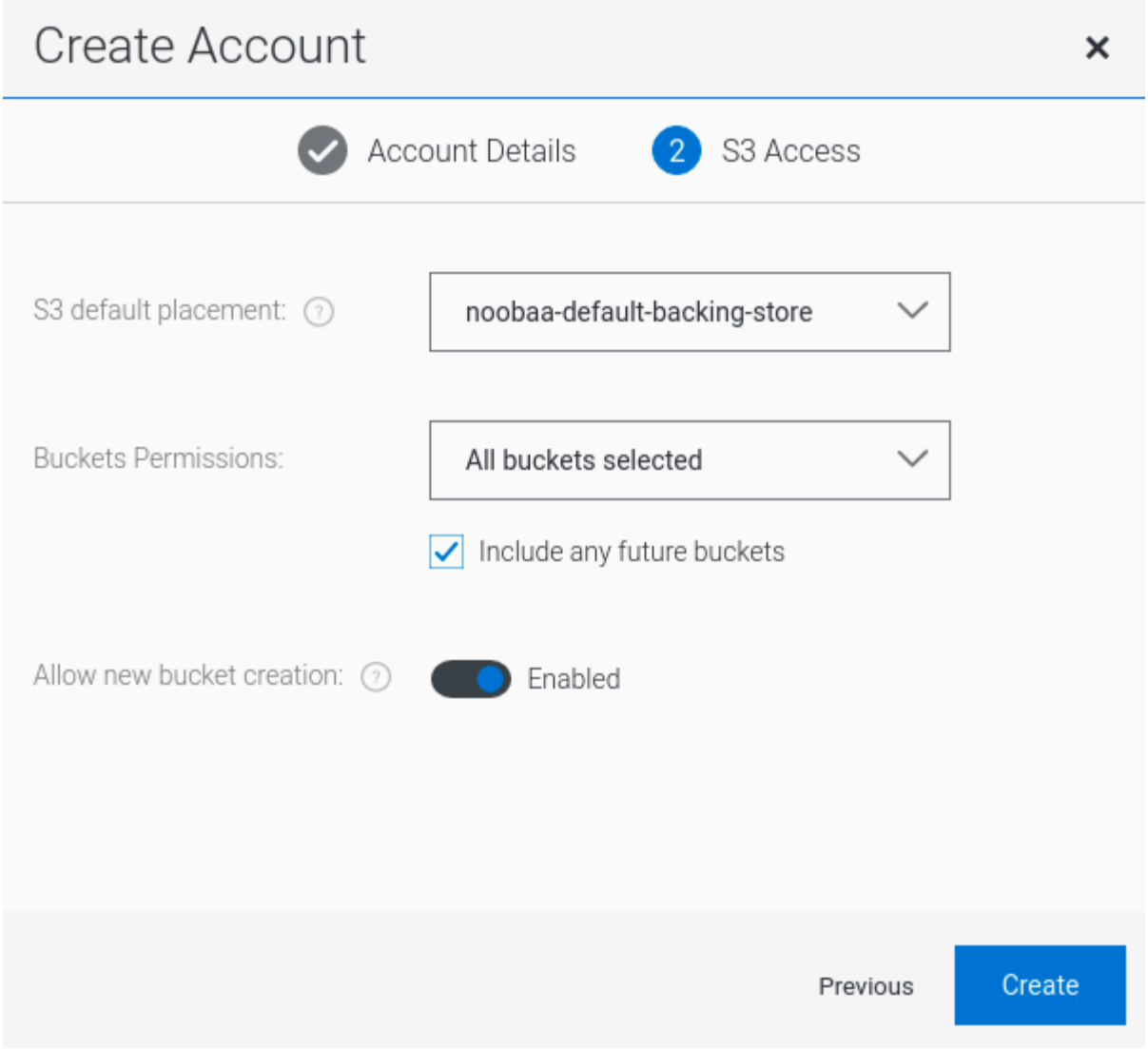


3.

S3 액세스만 선택하고 계정 이름 (예: john.doe@example.com)을 입력합니다 . 다음을 클릭합니다.



4. **S3 기본 배치 (예: noobaa-default-backing-store)를 선택합니다. 버킷 권한을 선택합니다. 특정 버킷 또는 모든 버킷을 선택할 수 있습니다. 생성을 클릭합니다.**


Create Account
Account Details
2
S3 Access
S3 default placement:
noobaa-default-backing-store
Buckets Permissions:
All buckets selected
Include any future buckets
Allow new bucket creation:
Enabled
Previous
Create

8장. 개체 버킷 클레임

개체 버킷 클레임은 워크로드에 대해 S3 호환 버킷 백엔드를 요청하는 데 사용할 수 있습니다.

다음 세 가지 방법으로 오브젝트 버킷 클레임을 생성할 수 있습니다.

- 8.1절. “동적 개체 버킷 클레임”
- 8.2절. “명령줄 인터페이스를 사용하여 개체 버킷 클레임 생성”
- 8.3절. “OpenShift 웹 콘솔을 사용하여 개체 버킷 클레임 생성”

개체 버킷 클레임은 새 액세스 키 및 시크릿 액세스 키를 포함하여 버킷에 대한 권한이 있는 NooBaa에 새 버킷 및 애플리케이션 계정을 생성합니다. 애플리케이션 계정은 단일 버킷에만 액세스할 수 있으며 기본적으로 새 버킷을 생성할 수 없습니다.

8.1. 동적 개체 버킷 클레임

영구 볼륨과 유사하게 애플리케이션의 YAML에 오브젝트 버킷 클레임의 세부 정보를 추가하고, 오브젝트 서비스 엔드포인트, 액세스 키 및 구성 맵 및 시크릿에서 사용할 수 있는 시크릿 액세스 키를 가져올 수 있습니다. 이 정보를 애플리케이션의 환경 변수로 동적으로 읽기가 쉽습니다.

절차

1. 애플리케이션 YAML에 다음 행을 추가합니다.

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: <obc-name>
spec:
  generateBucketName: <obc-bucket-name>
  storageClassName: openshift-storage.noobaa.io
```

이러한 행은 개체 버킷 클레임 자체입니다.

a. **<obc-name>** 을 고유한 개체 버킷 클레임 이름으로 바꿉니다.

b. **<obc-bucket-name>** 을 개체 버킷 클레임의 고유한 버킷 이름으로 바꿉니다.

2. **YAML** 파일에 더 많은 줄을 추가하여 개체 버킷 클레임의 사용을 자동화할 수 있습니다. 아래 예제는 데이터를 사용한 구성 맵과 인증 정보가 있는 시크릿인 버킷 클레임 결과 간의 매핑입니다. 이 특정 작업은 **NooBaa**의 개체 버킷을 클레임하여 버킷과 계정을 만듭니다.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
        - image: <your application image>
          name: test
          env:
            - name: BUCKET_NAME
              valueFrom:
                configMapKeyRef:
                  name: <obc-name>
                  key: BUCKET_NAME
            - name: BUCKET_HOST
              valueFrom:
                configMapKeyRef:
                  name: <obc-name>
                  key: BUCKET_HOST
            - name: BUCKET_PORT
              valueFrom:
                configMapKeyRef:
                  name: <obc-name>
                  key: BUCKET_PORT
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: <obc-name>
                  key: AWS_ACCESS_KEY_ID
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: <obc-name>
                  key: AWS_SECRET_ACCESS_KEY
```

a. **<obc-name>**의 모든 인스턴스를 오브젝트 버킷 클레임 이름으로 바꿉니다.

b. **<your 애플리케이션 이미지>를 애플리케이션 이미지로 바꿉니다.**

3. **업데이트된 YAML 파일을 적용합니다.**

```
# oc apply -f <yaml.file>
```

a. **<yaml.file> 을 YAML 파일의 이름으로 바꿉니다.**

4. **새 구성 맵을 보려면 다음을 실행합니다.**

```
# oc get cm <obc-name>
```

a. **obc-name 을 개체 버킷 클레임의 이름으로 바꿉니다.**

출력에 다음 환경 변수가 있을 수 있습니다.

- **BUCKET_HOST - 애플리케이션에서 사용할 엔드 포인트**
- **BUCKET_PORT - 애플리케이션에 사용할 수 있는 포트**
  - **포트는 BUCKET_HOST 와 관련이 있습니다. 예를 들어 BUCKET_HOST 가 https://my.example.com 이고 BUCKET_PORT 가 443이면 개체 서비스의 엔드포인트는 https://my.example.com:443 입니다.**
- **BUCKET_NAME - 요청되거나 생성된 버킷 이름**
- **AWS_ACCESS_KEY_ID - 인증 정보의 일부인 액세스 키**
- **AWS_SECRET_ACCESS_KEY - 인증 정보의 일부인 비밀 액세스 키**

중요

AWS_ACCESS_KEY_ID 및 **AWS_SECRET_ACCESS_KEY** 를 검색합니다. 이름은 **AWS S3 API**와 호환되도록 사용됩니다. 특히 **MCG(Multicloud Object Gateway)** 버킷에서 **S3** 작업을 수행하는 동안 키를 지정해야 합니다. 키는 **Base64**로 인코딩됩니다. 키를 사용하기 전에 디코딩합니다.

```
# oc get secret <obc_name> -o yaml
```

<obc_name>

오브젝트 버킷 클레임의 이름을 지정합니다.

8.2. 명령줄 인터페이스를 사용하여 개체 버킷 클레임 생성

명령줄 인터페이스를 사용하여 오브젝트 버킷 클레임을 생성할 때 구성 맵과 시크릿이 함께 제공됩니다. 이 시크릿에는 애플리케이션에서 오브젝트 스토리지 서비스를 사용하는 데 필요한 모든 정보가 포함됩니다.

사전 요구 사항

- **MCG** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

- **IBM Power Systems**의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

절차

1. 명령줄 인터페이스를 사용하여 새 버킷 및 자격 증명의 세부 정보를 생성합니다. 다음 명령을 실행합니다.

```
# noobaa obc create <obc-name> -n openshift-storage
```

<obc-name> 을 고유한 개체 버킷 클레임 이름으로 바꿉니다(예: **myappobc**).

또한 **--app-namespace** 옵션을 사용하여 오브젝트 버킷 클레임 구성 맵과 시크릿이 생성되는 네임스페이스(예: **myapp-namespace**)를 지정할 수 있습니다.

출력 예:

```
INFO[0001]   Created: ObjectBucketClaim "test21obc"
```

MCG 명령줄-인터페이스에서 필요한 구성을 생성하고 **OpenShift**에 새 **OBC**에 대해 알립니다.

2. 다음 명령을 실행하여 개체 버킷 클레임을 확인합니다.

```
# oc get obc -n openshift-storage
```

출력 예:

```
NAME       STORAGE-CLASS               PHASE   AGE
test21obc  openshift-storage.noobaa.io  Bound   38s
```

3. 다음 명령을 실행하여 새 오브젝트 버킷 클레임에 대한 **YAML** 파일을 확인합니다.

```
# oc get obc test21obc -o yaml -n openshift-storage
```

출력 예:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
```

```
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  generation: 2
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  resourceVersion: "40756"
  selfLink: /apis/objectbucket.io/v1alpha1/namespaces/openshift-
storage/objectbucketclaims/test21obc
  uid: 64f04cba-f662-11e9-bc3c-0295250841af
spec:
  ObjectBucketName: obc-openshift-storage-test21obc
  bucketName: test21obc-933348a6-e267-4f82-82f1-e59bf4fe3bb4
  generateBucketName: test21obc
  storageClassName: openshift-storage.noobaa.io
status:
  phase: Bound
```

4. **openshift-storage 네임스페이스 내에서 이 오브젝트 버킷 클레임을 사용하는 구성 맵과 시크릿을 찾을 수 있습니다. CM 및 시크릿의 이름은 개체 버킷 클레임과 동일합니다. 시크릿을 보려면 다음을 수행합니다.**

```
# oc get -n openshift-storage secret test21obc -o yaml
```

출력 예:

```
Example output:
apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: c0M0R2xVanF3ODR3bHBkVW94cmY=
  AWS_SECRET_ACCESS_KEY:
Wi9kcFluSWxHRzlWaFlzNk1hc0xma2JXcjM1MVhqa051SlBleXpmOQ==
kind: Secret
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  ownerReferences:
  - apiVersion: objectbucket.io/v1alpha1
    blockOwnerDeletion: true
```

```
    controller: true
    kind: ObjectBucketClaim
    name: test21obc
    uid: 64f04cba-f662-11e9-bc3c-0295250841af
  resourceVersion: "40751"
  selfLink: /api/v1/namespaces/openshift-storage/secrets/test21obc
  uid: 65117c1c-f662-11e9-9094-0a5305de57bb
type: Opaque
```

시크릿은 **S3** 액세스 자격 증명을 제공합니다.

5. 구성 맵을 보려면 다음을 수행합니다.

```
# oc get -n openshift-storage cm test21obc -o yaml
```

출력 예:

```
apiVersion: v1
data:
  BUCKET_HOST: 10.0.171.35
  BUCKET_NAME: test21obc-933348a6-e267-4f82-82f1-e59bf4fe3bb4
  BUCKET_PORT: "31242"
  BUCKET_REGION: ""
  BUCKET_SUBREGION: ""
kind: ConfigMap
metadata:
  creationTimestamp: "2019-10-24T13:30:07Z"
  finalizers:
  - objectbucket.io/finalizer
  labels:
    app: noobaa
    bucket-provisioner: openshift-storage.noobaa.io-obc
    noobaa-domain: openshift-storage.noobaa.io
  name: test21obc
  namespace: openshift-storage
  ownerReferences:
  - apiVersion: objectbucket.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ObjectBucketClaim
    name: test21obc
    uid: 64f04cba-f662-11e9-bc3c-0295250841af
  resourceVersion: "40752"
  selfLink: /api/v1/namespaces/openshift-storage/configmaps/test21obc
  uid: 651c6501-f662-11e9-9094-0a5305de57bb
```

구성 맵에는 애플리케이션의 **S3** 끝점 정보가 포함되어 있습니다.

8.3. OPENSHIFT 웹 콘솔을 사용하여 개체 버킷 클레임 생성

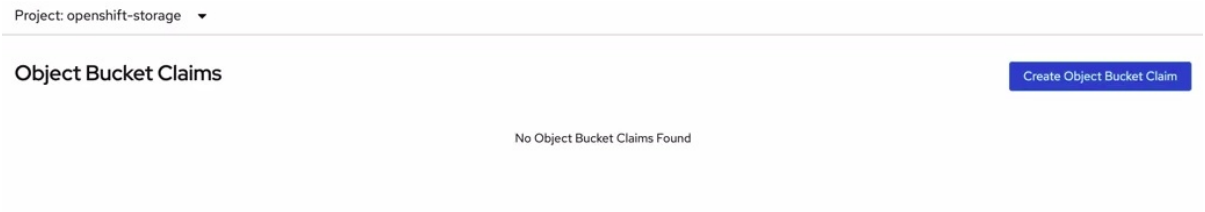
OpenShift 웹 콘솔을 사용하여 OBC(오브젝트 버킷 클레임)를 생성할 수 있습니다.

사전 요구 사항

- OpenShift 웹 콘솔에 대한 관리 액세스.
- 애플리케이션이 OBC와 통신하려면 configmap 및 secret을 사용해야 합니다. 이에 대한 자세한 내용은 8.1절. "동적 개체 버킷 클레임" 의 내용을 참조하십시오.

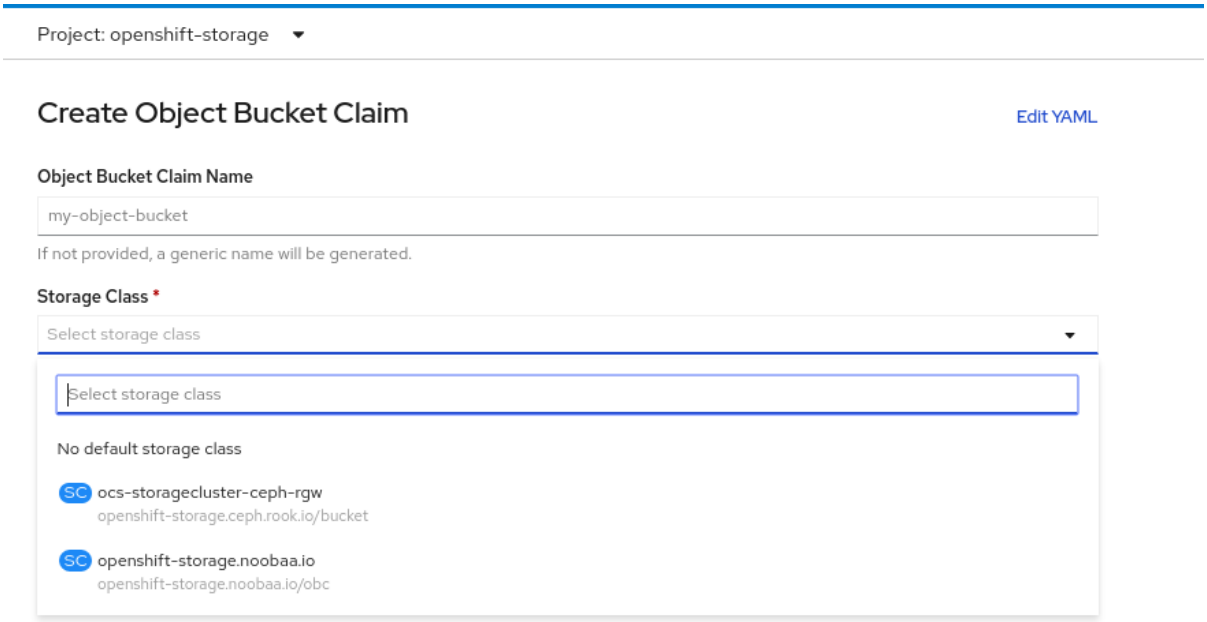
절차

1. OpenShift 웹 콘솔에 로그인합니다.
2. 왼쪽 네비게이션 바에서 스토리지 → 개체 버킷 클레임 을 클릭합니다.
3. 개체 버킷 클레임 생성을 클릭합니다.



4. 오브젝트 버킷 클레임의 이름을 입력하고 드롭다운 메뉴에서 배포(내부 또는 외부)를 기반으로 적절한 스토리지 클래스를 선택합니다.

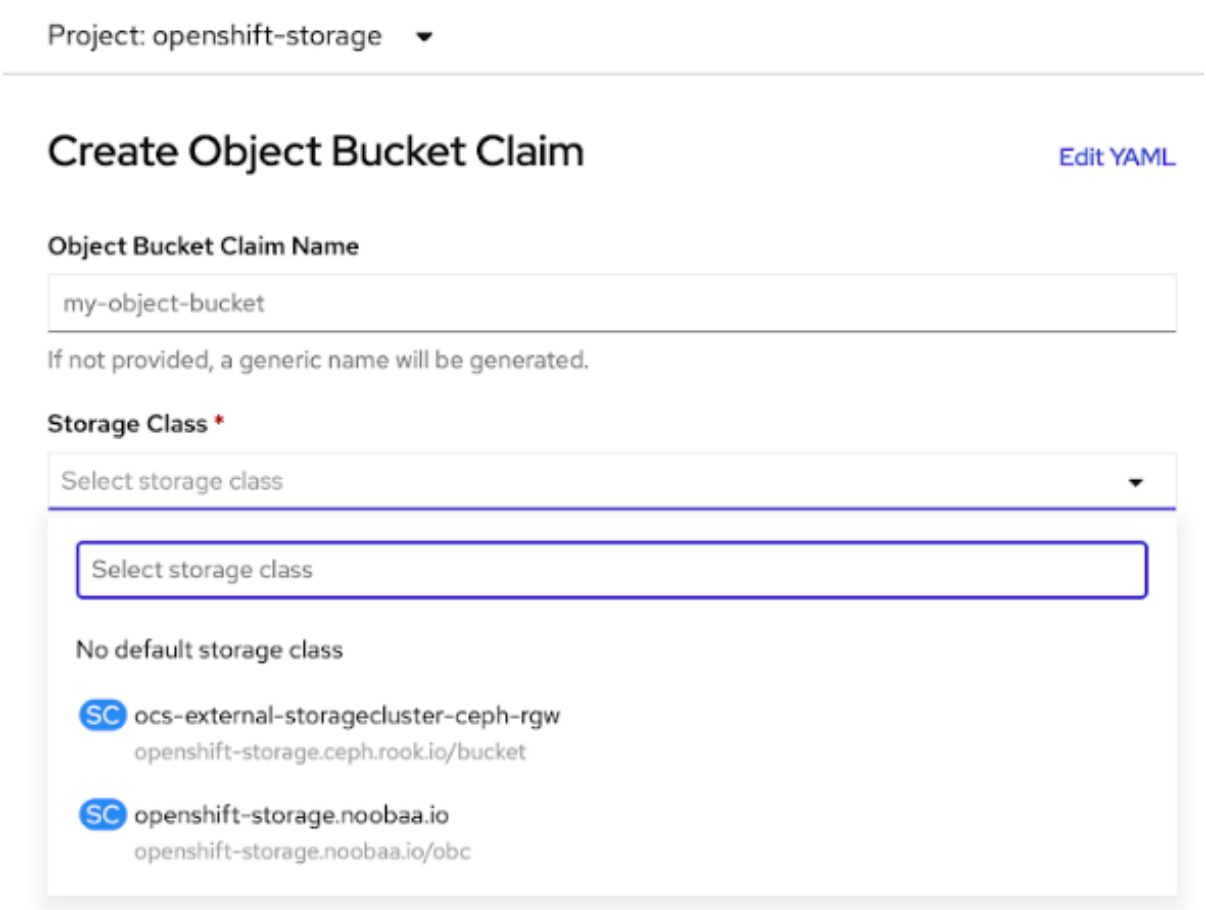
   내부 모드



배포 후에 생성된 다음 스토리지 클래스를 사용할 수 있습니다.

- **oCS-storagecluster-ceph-rgw** 는 **Ceph Object Gateway(RGW)**를 사용합니다.
- **openshift-storage.noobaa.io** 는 **Multicloud Object Gateway**를 사용합니다.

외부 모드



배포 후에 생성된 다음 스토리지 클래스를 사용할 수 있습니다.

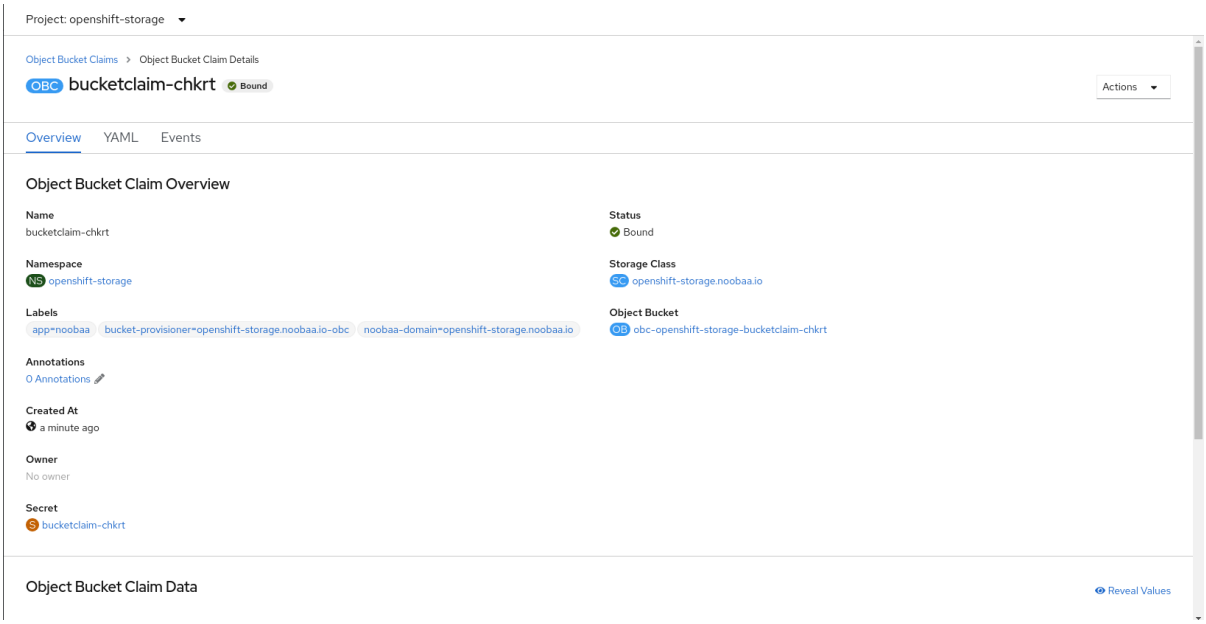
- **OCS-external-storagecluster-ceph-rgw** 는 **Ceph RGW(**오브젝트 게이트웨이**)**를 사용합니다.
- **openshift-storage.noobaa.io** 는 **Multicloud Object Gateway**를 사용합니다.

참고

RGW OBC 스토리지 클래스는 **OpenShift Container Storage** 버전 **4.5**의 새로운 설치에서만 사용할 수 있습니다. 이전 **OpenShift Container Storage** 릴리스에서 업그레이드된 클러스터에는 적용되지 않습니다.

5. 생성을 클릭합니다.

OBC를 생성하면 세부 정보 페이지로 리디렉션됩니다.



추가 리소스

- *8장. 개체 버킷 클레임*

8.4. 배포에 오브젝트 버킷 클레임 연결

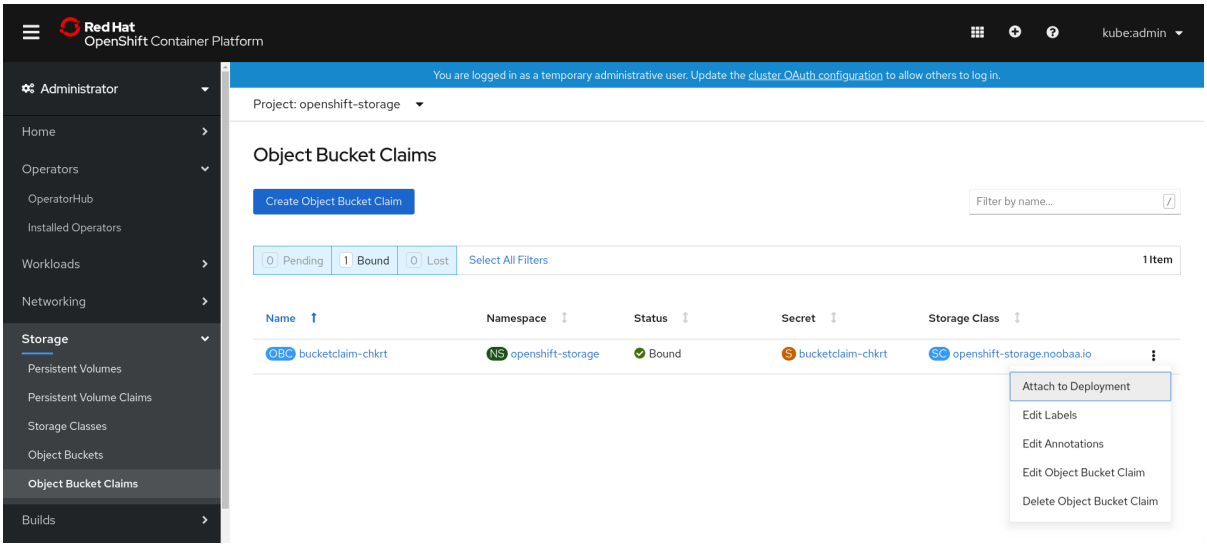
생성되고 나면 **OBC(오브젝트 버킷 클레임)**를 특정 배포에 연결할 수 있습니다.

사전 요구 사항

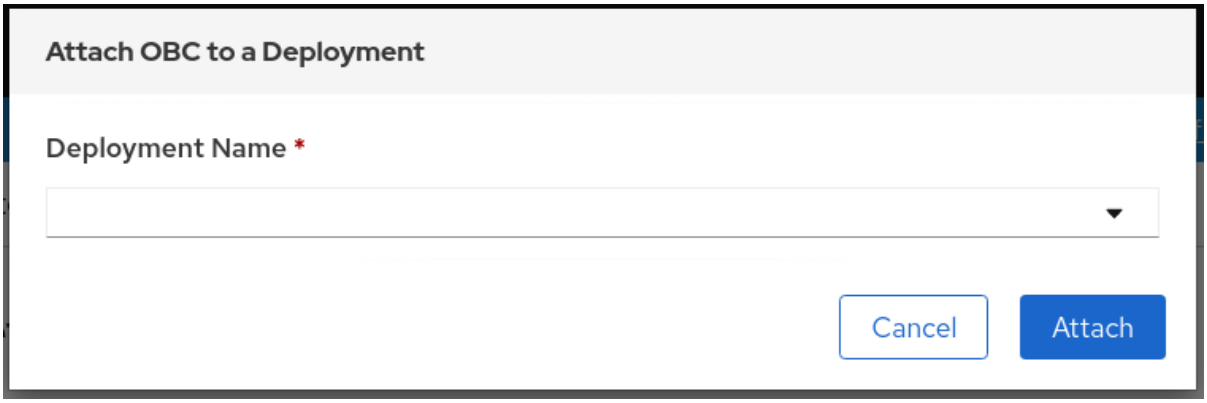
- **OpenShift** 웹 콘솔에 대한 관리 액세스.

절차

1. 왼쪽 네비게이션 바에서 스토리지 → 개체 버킷 클레임 을 클릭합니다.
2. 생성한 **OBC** 옆에 있는 작업 메뉴(**kube**)를 클릭합니다.
3. 드롭다운 메뉴에서 **Attach to Deployment** (배포에 연결)를 선택합니다.



4. **Deployment Name(배포 이름) 목록에서 원하는 배포를 선택한 다음 Attach:**



추가 리소스

- *8장. 개체 버킷 클레임*

8.5. OPENSHIFT 웹 콘솔을 사용하여 오브젝트 버킷 보기

OpenShift 웹 콘솔을 사용하여 OBC(오브젝트 버킷 클레임)에 대해 생성된 개체 버킷의 세부 정보를 볼 수 있습니다.

사전 요구 사항

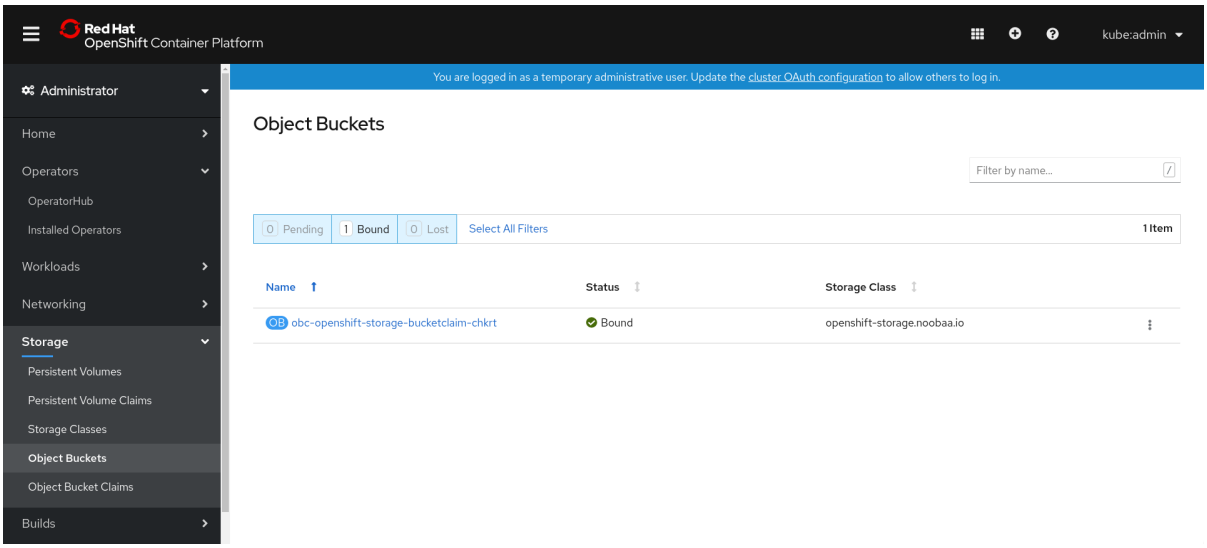
- **OpenShift 웹 콘솔에 대한 관리 액세스.**

절차

오브젝트 버킷 세부 정보를 보려면 다음을 수행합니다.

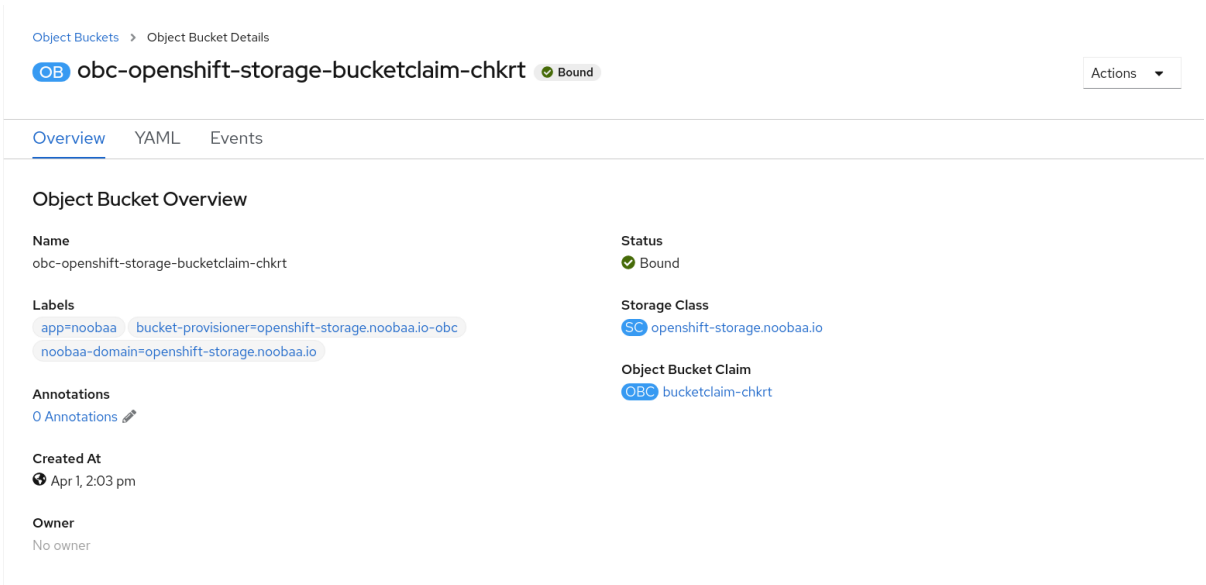
1. **OpenShift 웹 콘솔에 로그인합니다.**

2. **왼쪽 네비게이션 바에서 스토리지 → 개체 버킷을 클릭합니다.**



특정 OBC의 세부 정보 페이지로 이동하고 리소스 링크를 클릭하여 해당 OBC의 개체 버킷을 볼 수도 있습니다.

3. **세부 정보를 볼 개체 버킷을 선택합니다. 오브젝트 버킷의 세부 정보 페이지로 이동합니다.**



추가 리소스

- *8장. 개체 버킷 클레임*

8.6. 개체 버킷 클레임 삭제

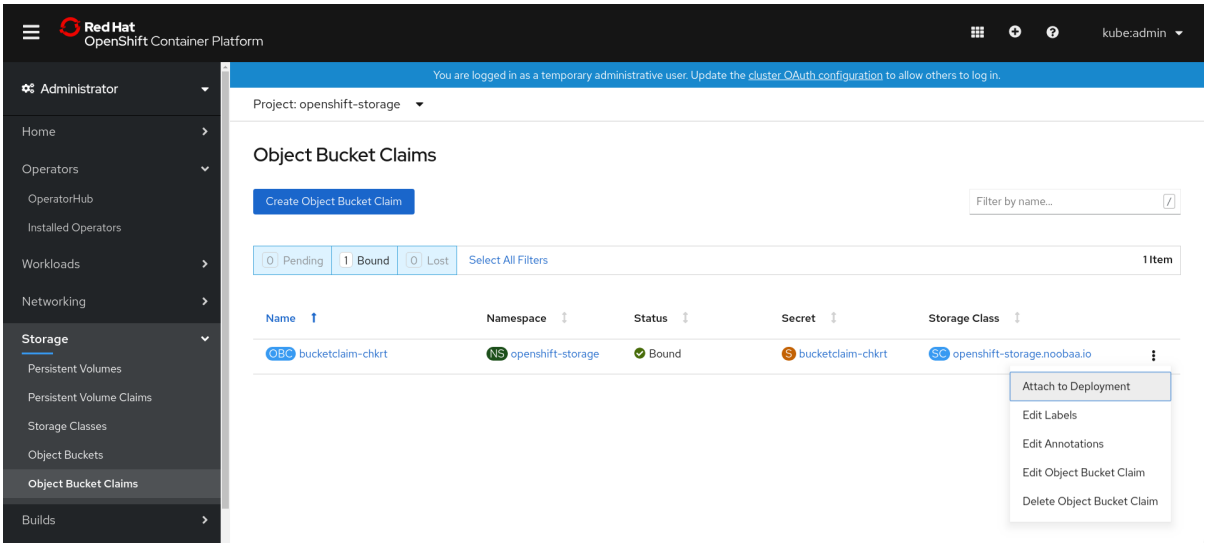
사전 요구 사항

- **OpenShift** 웹 콘솔에 대한 관리 액세스.

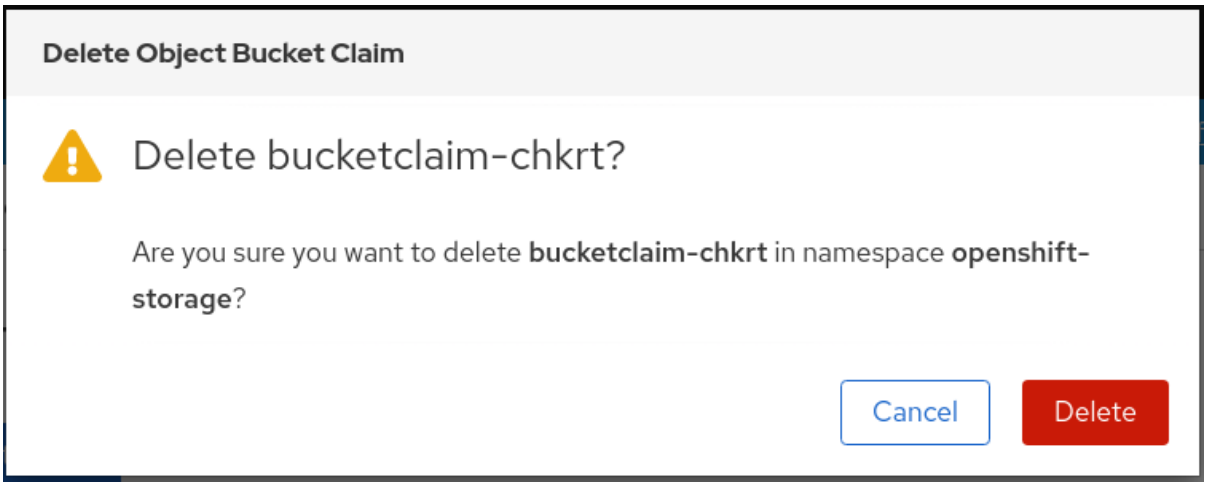
절차

1. 왼쪽 네비게이션 바에서 스토리지 → 개체 버킷 클레임 을 클릭합니다.

2. 삭제할 개체 버킷 클레임 옆에 있는 작업 메뉴(**kube**)를 클릭합니다.



3. 메뉴에서 개체 버킷 클레임 삭제 를 선택합니다.



4. 삭제를 클릭합니다.

추가 리소스

- 8장. 개체 버킷 클레임

9장. 오브젝트 버킷의 캐싱 정책

캐시 버킷은 **hub** 대상 및 캐시 대상이 있는 네임스페이스 버킷입니다. **hub** 대상은 **S3** 호환 가능한 대형 오브젝트 스토리지 버킷입니다. 캐시 버킷은 로컬 **Multicloud Object Gateway** 버킷입니다. **AWS** 버킷 또는 **IBM COS** 버킷을 캐시하는 캐시 버킷을 생성할 수 있습니다.

- AWS S3
- IBM COS

9.1. AWS 캐시 버킷 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들어 **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package** 에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.

참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1. 네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object**

Gateway 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa namespacestore create aws-s3 <namespacestore> --access-key <AWS ACCESS KEY> --secret-key <AWS SECRET ACCESS KEY> --target-bucket <bucket-name>
```

a. **<namespacestore>** 를 네임스페이스 저장소의 이름으로 바꿉니다.

b. **<AWS ACCESS KEY>** 및 **<AWS SECRET ACCESS KEY>** 를 이 목적을 위해 생성한 **AWS** 액세스 키 **ID** 및 시크릿 액세스 키로 바꿉니다.

c. **<bucket-name>** 을 기존 **AWS** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

YAML을 적용하여 스토리지 리소스를 추가할 수도 있습니다. 먼저 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: <namespacestore-secret-name>
type: Opaque
data:
  AWS_ACCESS_KEY_ID: <AWS ACCESS KEY ID ENCODED IN BASE64>
  AWS_SECRET_ACCESS_KEY: <AWS SECRET ACCESS KEY ENCODED IN BASE64>
```

Base64를 사용하여 자체 **AWS** 액세스 키 **ID** 및 시크릿 액세스 키를 제공하고 인코딩해야 하며 **<AWS ACCESS KEY ID ENCODED in BASE64>** 및 **< AWS SECRET ACCESS KEY ENCODED in BASE64>** 대신 결과를 사용해야 합니다.

<namespacestore-secret-name> 을 고유한 이름으로 바꿉니다.

그런 다음 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
```

```
  labels:
    app: noobaa
  name: <namespacestore>
  namespace: openshift-storage
spec:
  awsS3:
    secret:
      name: <namespacestore-secret-name>
      namespace: <namespace-secret>
    targetBucket: <target-bucket>
  type: aws-s3
```

d. **<namespacestore>** 를 고유한 이름으로 바꿉니다.

e. **<namespacestore-secret-name>** 을 이전 단계에서 생성한 보안으로 바꿉니다.

f. **<namespace-secret>** 을 이전 단계에서 보안을 생성하는 데 사용된 네임스페이스로 바꿉니다.

g. **<target-bucket>** 을 네임스페이스 저장소용으로 생성한 **AWS S3** 버킷으로 바꿉니다.

2. 다음 명령을 실행하여 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass cache <my-cache-bucket-class> --
backingstores <backing-store> --hub-resource <namespacestore>
```

a. **<my-cache-bucket-class>** 를 고유한 버킷 클래스 이름으로 바꿉니다.

b. **<backing-store>** 를 관련 백업 저장소로 바꿉니다. 이 필드에 쉼표로 구분된 하나 이상의 백업 저장소를 나열할 수 있습니다.

c. **<namespacestore>** 를 이전 단계에서 생성한 네임스페이스 저장소로 바꿉니다.

3. 다음 명령을 실행하여 **2**단계에 정의된 버킷 클래스를 사용하는 개체 버킷 클레임 리소스를 사용하여 버킷을 생성합니다.

```
noobaa obc create <my-bucket-claim> my-app --bucketclass <custom-bucket-class>
```

a. **<my-bucket-claim>** 을 고유한 이름으로 바꿉니다.

b. **<custom-bucket-class>** 를 2단계에서 생성된 버킷 클래스의 이름으로 바꿉니다.

9.2. IBM COS 캐시 버킷 생성

사전 요구 사항

- **MCG(Multicloud Object Gateway)** 명령줄 인터페이스를 다운로드합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-x86_64-rpms
# yum install mcg
```

참고

서브스크립션 관리자를 사용하여 리포지토리를 활성화하기 위한 적절한 아키텍처를 지정합니다. 예를 들면 다음과 같습니다.

- **IBM Power Systems**의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-ppc64le-rpms
```

- **IBM Z** 인프라의 경우 다음 명령을 사용합니다.

```
# subscription-manager repos --enable=rh-ocs-4-for-rhel-8-s390x-rpms
```

또는 **https://access.redhat.com/downloads/content/547/ver=4/rhel---8/4/x86_64/package** 에 있는 **OpenShift Container Storage RPM**에서 **mcg** 패키지를 설치할 수도 있습니다.

참고

아키텍처에 따라 올바른 제품 변형을 선택합니다.

절차

1.

네임스페이스 저장소 리소스를 생성합니다. 네임 스페이스 저장소는 **Multicloud Object Gateway** 네임 스페이스 버킷의 데이터에 대한 읽기 또는 쓰기 대상으로 사용할 기본 스토리지를 나타냅니다. **MCG** 명령줄 인터페이스에서 다음 명령을 실행합니다.

```
noobaa namespacestore create ibm-cos <namespacestore> --endpoint <IBM COS
ENDPOINT> --access-key <IBM ACCESS KEY> --secret-key <IBM SECRET ACCESS
KEY> --target-bucket <bucket-name>
```

a.

<namespacestore> 를 네임 스페이스 저장소의 이름으로 바꿉니다.

b.

<IBM ACCESS KEY>, <IBM SECRET ACCESS KEY>, <IBM COS ENDPOINT> 를 **IBM** 액세스 키 **ID,** 시크릿 액세스 키 및 기존 **IBM** 버킷 위치에 해당하는 해당 지역 엔드포인트로 바꿉니다.

c.

<bucket-name> 을 기존 **IBM** 버킷 이름으로 바꿉니다. 이 인수는 **Multicloud Object Gateway**에 백업 저장소에 대상 버킷으로 사용할 버킷과 그 다음에는 데이터 스토리지 및 관리에 지시합니다.

YAML을 적용하여 스토리지 리소스를 추가할 수도 있습니다. 먼저 인증 정보를 사용하여 보안을 생성합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: <namespacestore-secret-name>
type: Opaque
data:
  IBM_COS_ACCESS_KEY_ID: <IBM COS ACCESS KEY ID ENCODED IN BASE64>
  IBM_COS_SECRET_ACCESS_KEY: <IBM COS SECRET ACCESS KEY ENCODED
IN BASE64>
```

Base64를 사용하여 자체 **IBM COS** 액세스 키 **ID** 및 비밀 액세스 키를 제공하고 인코딩해야 하며 **<IBM COS COS** 키 **ID ENCODED IN BASE64>** 및 **< IBM COS SECRET ACCESS KEY ENCODED IN BASE64>'** 대신 결과를 사용해야 합니다.

<namespacestore-secret-name> 을 고유한 이름으로 바꿉니다.

그런 다음 다음 **YAML**을 적용합니다.

```
apiVersion: noobaa.io/v1alpha1
```

```
kind: NamespaceStore
metadata:
  finalizers:
  - noobaa.io/finalizer
  labels:
    app: noobaa
  name: <namespacestore>
  namespace: openshift-storage
spec:
  s3Compatible:
    endpoint: <IBM COS ENDPOINT>
    secret:
      name: <backingstore-secret-name>
      namespace: <namespace-secret>
    signatureVersion: v2
    targetBucket: <target-bucket>
  type: ibm-cos
```

d. **<namespacestore>** 를 고유한 이름으로 바꿉니다.

e. **<IBM COS ENDPOINT>** 를 적절한 **IBM COS** 엔드포인트로 바꿉니다.

f. **<backingstore-secret-name>** 을 이전 단계에서 생성한 보안으로 바꿉니다.

g. **<namespace-secret>** 을 이전 단계에서 보안을 생성하는 데 사용된 네임스페이스로 바꿉니다.

h. **<target-bucket>** 을 네임스페이스 저장소용으로 생성한 **AWS S3** 버킷으로 바꿉니다.

2. 다음 명령을 실행하여 버킷 클래스를 생성합니다.

```
noobaa bucketclass create namespace-bucketclass cache <my-bucket-class> --
backingstores <backing-store> --hubResource <namespacestore>
```

a. **<my-bucket-class>** 를 고유한 버킷 클래스 이름으로 바꿉니다.

b. **<backing-store>** 를 관련 백업 저장소로 바꿉니다. 이 필드에 쉼표로 구분된 하나 이상의 백업 저장소를 나열할 수 있습니다.

c.

<namespacestore> 를 이전 단계에서 생성한 네임스페이스 저장소로 바꿉니다.

3.

다음 명령을 실행하여 **2**단계에 정의된 버킷 클래스를 사용하는 개체 버킷 클레임 리소스를 사용하여 버킷을 생성합니다.

```
noobaa obc create <my-bucket-claim> my-app --bucketclass <custom-bucket-class>
```

a.

<my-bucket-claim> 을 고유한 이름으로 바꿉니다.

b.

<custom-bucket-class> 를 **2**단계에서 생성된 버킷 클래스의 이름으로 바꿉니다.

10장. 끝점을 추가하여 MULTICLOUD OBJECT GATEWAY 성능 확장

Multicloud Object Gateway 성능은 환경마다 다를 수 있습니다. 특정 애플리케이션에는 기술 프리뷰 기능인 S3 엔드포인트를 확장하여 쉽게 해결할 수 있는 빠른 성능이 필요한 경우도 있습니다.

Multicloud Object Gateway 리소스 풀은 기본적으로 두 가지 유형의 서비스를 제공하는 NooBaa 데몬 컨테이너 그룹입니다.

- 스토리지 서비스
- S3 끝점 서비스

중요

엔드포인트를 추가하여 Multicloud Object Gateway 성능 확장은 기술 프리뷰 기능입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

10.1. MULTICLOUD OBJECT GATEWAY의 S3 끝점

S3 엔드포인트는 모든 Multicloud Object Gateway가 Multicloud Object Gateway에서 데이터 다이제스트를 처리하는 데 기본적으로 제공하는 서비스입니다. 엔드포인트 서비스는 인라인 데이터 청크, 중복 제거, 압축 및 암호화를 처리하고 Multicloud Object Gateway의 데이터 배치 지침을 허용합니다.

10.2. 스토리지 노드를 사용한 확장

사전 요구 사항

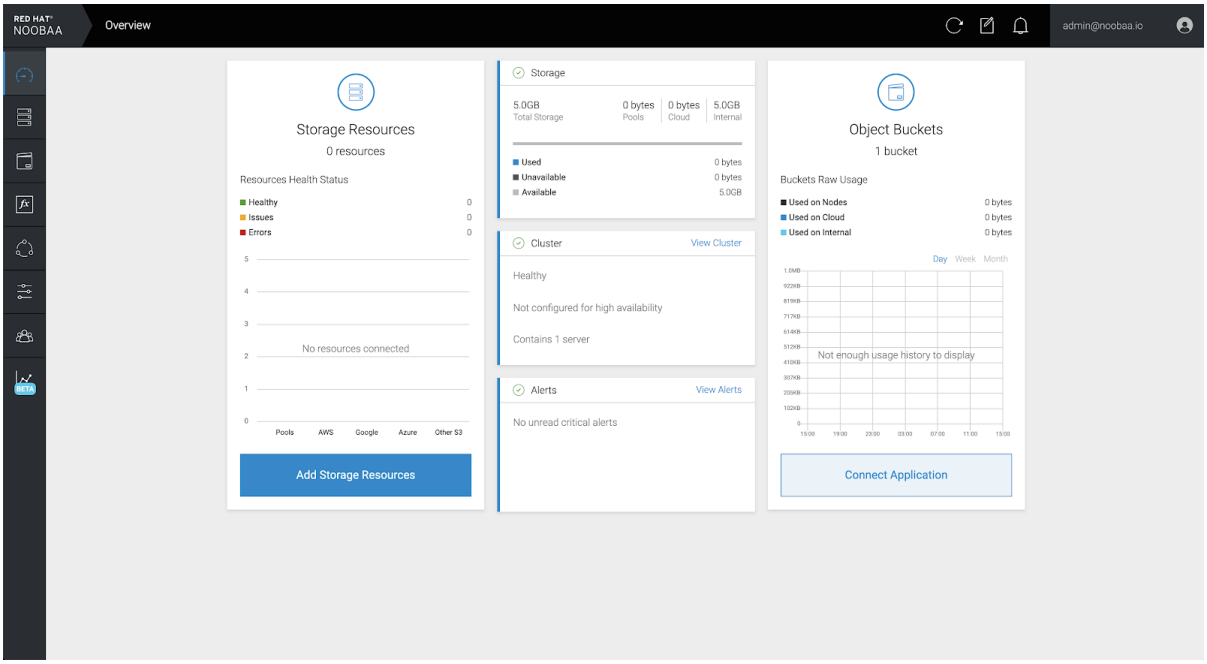
- Multicloud Object Gateway에 액세스할 수 있는 OpenShift Container Platform에서 실행 중인 OpenShift Container Storage 클러스터입니다.

Multicloud Object Gateway의 스토리지 노드는 하나 이상의 영구 볼륨에 연결되어 로컬 오브젝트 서비스 데이터 스토리지에 사용되는 NooBaa 데몬 컨테이너입니다. NooBaa 데몬은 Kubernetes 노드에

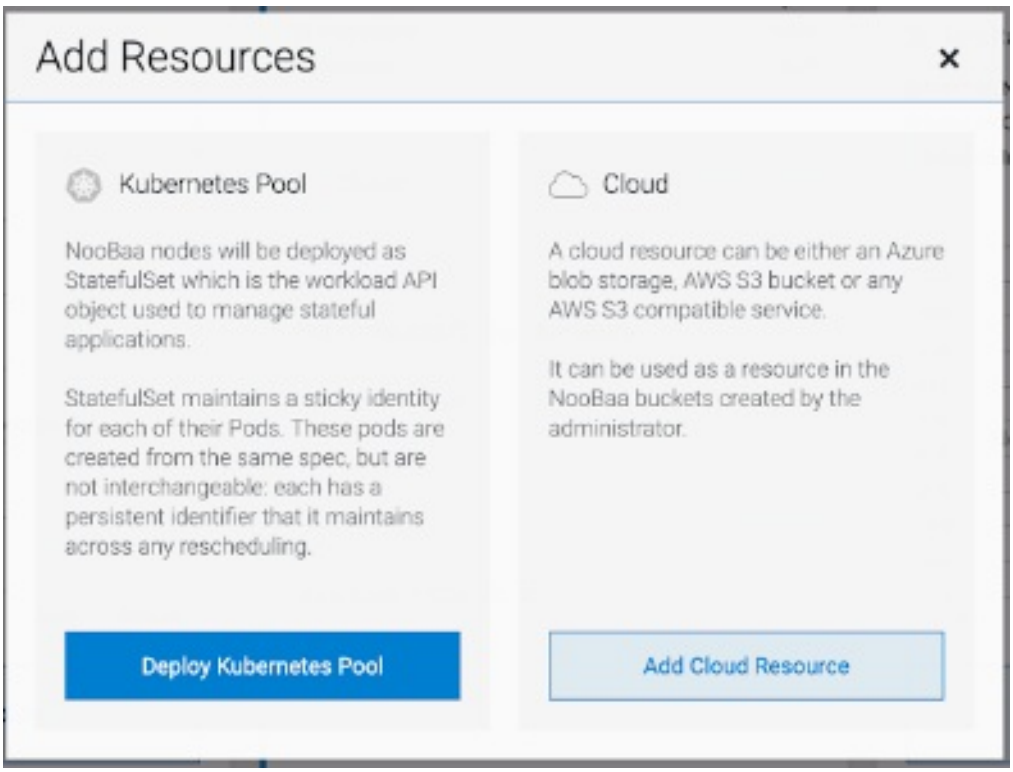
배포할 수 있습니다. 이 작업은 **StatefulSet Pod**로 구성된 **Kubernetes** 풀을 생성하여 수행할 수 있습니다.

절차

1. **Multicloud Object Gateway** 사용자 인터페이스의 **Overview** (개요) 페이지에서 **Add Storage Resources**(스토리지 리소스 추가)를 클릭합니다.

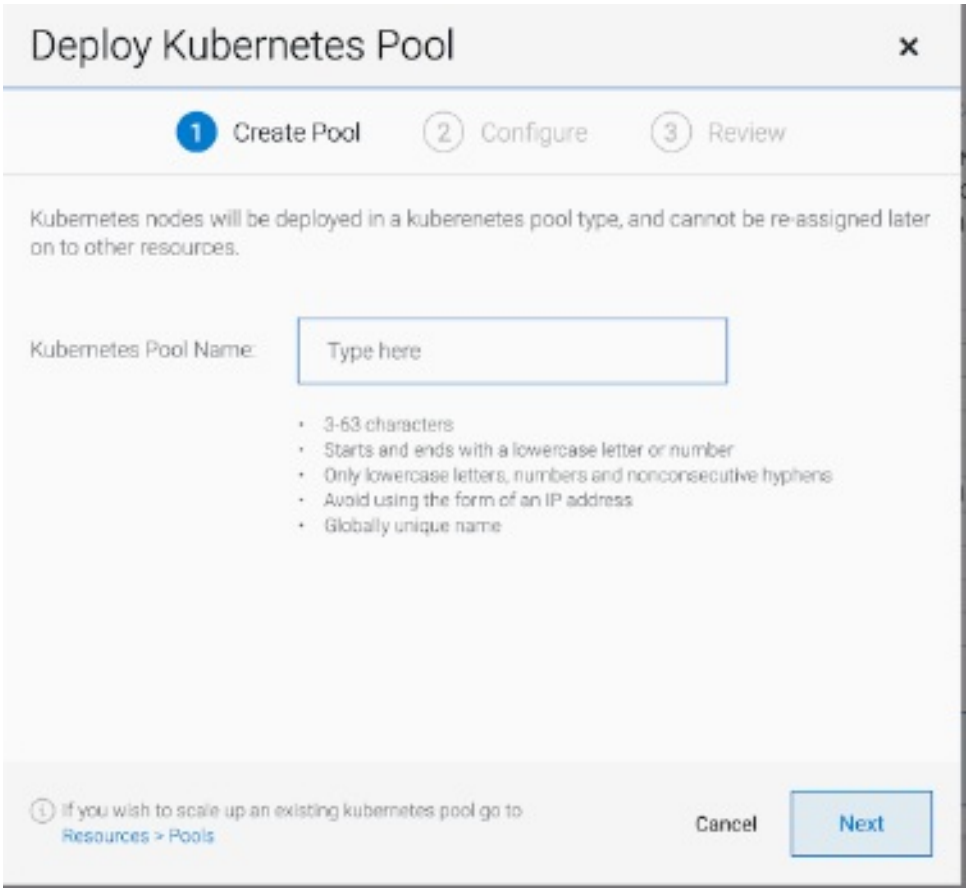


2. 창에서 **Deploy Kubernetes Pool (Kubernetes** 풀 배포)을 클릭합니다.



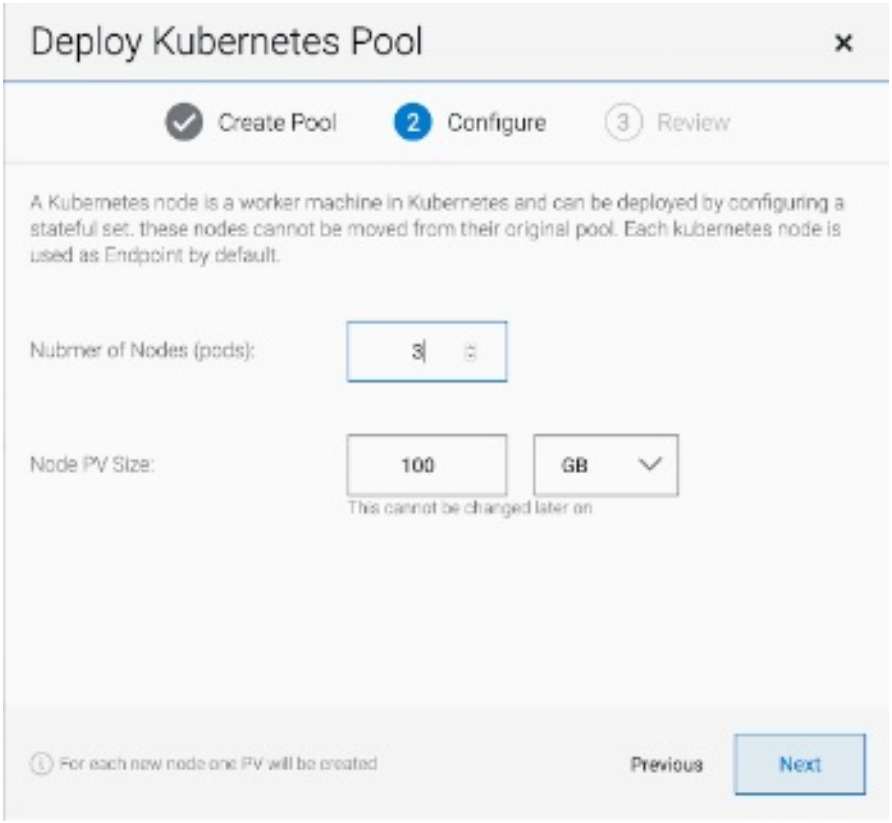
3.

Create Pool(풀 만들기) 단계에서 향후 설치된 노드의 대상 풀을 생성합니다.



4.

구성 단계에서 요청된 Pod 수 및 각 PV 크기를 구성합니다. 새 포드마다 하나의 PV가 생성됩니다.

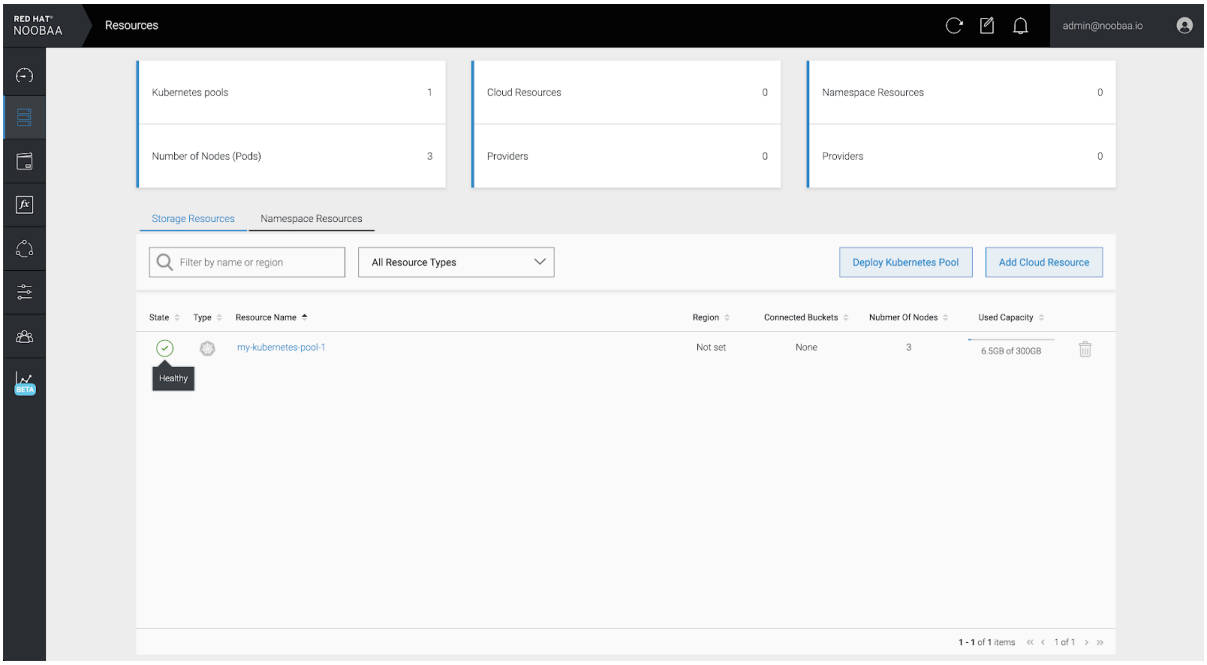


5.

검토 단계에서 새 풀의 세부 정보를 찾고 사용할 배포 방법을 선택할 수 있습니다(로컬 또는 외부 배포). 로컬 배포를 선택하면 **Kubernetes** 노드가 클러스터 내에 배포됩니다. 외부 배포를 선택하면 외부에서 실행할 **YAML** 파일이 제공됩니다.

6.

모든 노드는 첫 번째 단계에서 선택한 풀에 할당되며 리소스 → 스토리지 리소스 → 리소스 이름:



11장. MULTICLOUD OBJECT GATEWAY 엔드 포인트 자동 확장

MCG S3 서비스의 부하가 증가하거나 감소하면 MCG 개체 게이트웨이(MCG) 엔드포인트 수가 자동으로 확장됩니다. {product-name-short} 클러스터는 하나의 활성 MCG 끝점과 함께 배포됩니다. 각 MCG 끝점 Pod는 기본적으로 요청과 일치하는 제한이 있는 1개의 CPU 및 2Gi 메모리 요청으로 구성됩니다. 엔드포인트의 CPU 로드가 일관된 기간 동안 80%의 사용 임계값을 초과하면 첫 번째 엔드포인트의 부하를 줄이는 두 번째 엔드포인트가 배포됩니다. 두 끝점의 평균 CPU 로드가 일관된 기간 동안 80% 임계값 미만으로 떨어지면 엔드포인트 중 하나가 삭제됩니다. 이 기능을 사용하면 MCG의 성능 및 서비스 가능성이 향상됩니다.

12장. RADOS 오브젝트 게이트웨이 S3 끝점 액세스

사용자는 **RGW(RADOS Object Gateway)** 엔드포인트에 직접 액세스할 수 있습니다.

사전 요구 사항

- 실행 중인 **OpenShift Container Storage Platform**

절차

1. **oc get service** 명령을 실행하여 **RGW** 서비스 이름을 가져옵니다.

```
$ oc get service

NAME                                          TYPE
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   ClusterIP

CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
172.30.99.207    <none>        80/TCP     4d15h
```

2. **oc expose** 명령을 실행하여 **RGW** 서비스를 노출합니다.

```
$ oc expose svc/<RGW service name> --hostname=<route name>
```

<RGW-service name> 을 이전 단계의 **RGW** 서비스 이름으로 바꿉니다.

<route name> 을 **RGW** 서비스에 생성할 경로로 바꿉니다.

예를 들면 다음과 같습니다.

```
$ oc expose svc/rook-ceph-rgw-ocs-storagecluster-cephobjectstore --hostname=rook-ceph-rgw-ocs.ocp.host.example.com
```

3. **oc get route** 명령을 실행하여 **oc expose** 가 성공했으며 **RGW** 경로가 있는지 확인합니다.

```
$ oc get route

NAME                                          HOST/PORT                        PATH
```

```
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   rook-ceph-rgw-
ocsocp.host.example.com

SERVICES                                PORT        TERMINATION   WILDCARD
rook-ceph-rgw-ocs-storagecluster-cephobjectstore   http        <none>
```

확인

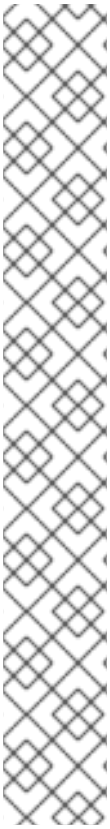
- **ENDPOINT** 를 확인하려면 다음 명령을 실행합니다.

```
aws s3 --no-verify-ssl --endpoint <ENDPOINT> ls
```

<ENDPOINT> 를 위의 **3**단계의 명령에서 가져오는 경로로 바꿉니다.

예를 들면 다음과 같습니다.

```
$ aws s3 --no-verify-ssl --endpoint http://rook-ceph-rgw-ocs.ocp.host.example.com ls
```



참고

기본 사용자 **ocs-storagecluster-cephobjectstoreuser** 의 액세스 키와 시크릿을 가져오려면 다음 명령을 실행합니다.

- 액세스 키:

```
$ oc get secret rook-ceph-object-user-ocs-storagecluster-cephobjectstore-ocs-
storagecluster-cephobjectstoreuser -n openshift-storage -o yaml | grep -w
"AccessKey:" | head -n1 | awk '{print $2}' | base64 --decode
```

- 시크릿 키:

```
$ oc get secret rook-ceph-object-user-ocs-storagecluster-cephobjectstore-ocs-
storagecluster-cephobjectstoreuser -n openshift-storage -o yaml | grep -w
"SecretKey:" | head -n1 | awk '{print $2}' | base64 --decode
```